National Défense
Defence nationale

MODIFIED HOPPING
SYNTHESIZER CONTROLLER (U)

by

Robin Addison

[DTIC QUALITY INSPECTED
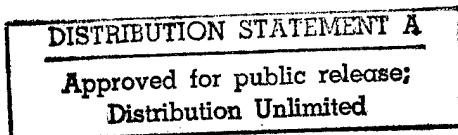
DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT NO. 1304

Canada

19970115 056

December 1996
Ottawa

National Défense
Defence nationale

MODIFIED HOPPING
SYNTHESIZER CONTROLLER (U)

by

Robin Addison
MILSATCOM, Space systems and Technology Section

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT NO. 1304

PROJECT
5CA11

December 1996
Ottawa

Abstract

Defence Research Establishment Ottawa is pursuing an in-house research activity in spread-spectrum technology to support development of robust, anti-jam satellite communications for the military. The in-house effort consists of developing a system simulator to research the techniques involved in spread-spectrum synchronization. For these experiments, a Hopping Synthesizer Controller is required to drive a frequency synthesizer, and is the subject of this report. The controller was built on a single board using a digital signal processor for calculations and an erasable programmable logic device for interfacing. It provides the pseudo-random hopping sequence, modulation, channel selection and Doppler precorrection. The Hopping Synthesizer Controller can drive the frequency synthesizer at hop rates up to 25 khops/s. Four configurations are supported: operation within either the payload or ground terminal simulator, for uplink or downlink. Each configuration supports commands from a Synchronization Controller and special uplink and downlink synchronization modes. In this document the software and hardware details are provided along with a user guide for the board.

Résumé

Le Centre de Recherche pour la Défense Ottawa, dans un projet de recherche interne, travaille au développement de technologies à spectre étalé reliées a un système de communications militaires par satellite robuste et à l'épreuve de l'interférence. Le groupe MILSATCOM (communications militaires par satellite) travaille sur certains aspects difficiles du projet comme la synchronisation. Pour ce travail, un contrôleur de synthétiseur de fréquences est requis pour chaque liaison, ascendante et descendante, ainsi qu'à chaque terminaison de ces liaisons. Ce contrôleur, construit sur une plaquette, est composé d'une unité de traitement de signaux numériques, d'une composante logique programmable et effaçable pour l'interface, et de composantes supportées. La modulation, la sélection du canal, le nombre pseudo-aléatoire pour choisir le bond ainsi que la correction Doppler sont tous utilisés pour le calcul des fréquences. Le contrôleur a deux modes spéciaux pour la synchronisation et peut recevoir des commandes par le contrôleur de synchronisation. Le contrôleur de synthétiseur de fréquences peut fonctionner jusqu'à 25 milles bonds de fréquence par seconde. Les détails du circuit et du logiciel sont inclus dans ce rapport ainsi qu'un guide d'utilisation.

Executive Summary

Defence Research Establishment Ottawa is pursuing an in-house research activity in spread-spectrum technology to support development of robust, anti-jam satellite communications for the military. The MILSATCOM (Military Satellite Communications) group is developing a system simulator to research some of the critical techniques such as spread-spectrum synchronization. Any work in this area requires a frequency hopping synthesizer such as a Comstron or Sciteq synthesizer, for each end of the link, that is under control of the Synchronization Controller. The synthesizers accept a frequency word from an external port and switch to this frequency. This report describes a Hopping Synthesizer Controller which generates this word at the appropriate time and also responds to commands from the Synchronization Controller. The Synchronization Controller is the processor that supervises all aspects of synchronization including adjusting internal clocks and directing the Hopping Synthesizer Controller.

The design of this Hopping Synthesizer Controller includes a digital signal processor to do the calculations and respond to external commands from the Synchronization Controller. The calculations involve producing the random hop frequency, adding frequency-shift-keying (FSK) modulation, channel selection, precorrecting the Doppler offsets and frequency word format conversion for the hopping synthesizer. The board supports the Synchronization Controller by precomputing the synchronization hop and probe frequencies and rapidly switching them on demand. The Synchronization Controller can override any of the default parameters of the frequency calculation. The board is configurable for use in uplink or downlink mode and for either ground terminal or satellite operation.

Flexibility was a key aspect of the board design. This board can be used in any of the four locations described previously that require a synthesizer. The use of a digital signal processor and programmable read-only memories permits simple modification of software algorithms such as random number generation, frequency calculations or synthesizer frequency word formatting. The use of an erasable programmable logic device (EPLD) allows simple and rapid modification of the hardware interface.

The EPLDs were found to be very useful in replacing the numerous small integrated circuits normally required for any design. The design and programming are done on a personal computer thus facilitating alterations within the EPLD. Substituting one EPLD for all the usual support integrated circuits meant rapid development of the hardware as well as easy customization of the hardware interface which simplified the software requirement.

The Hopping Synthesizer Controller was designed and built to drive a Comstron or Sciteq synthesizer in a frequency hopped communications system. This was accomplished with the board successfully tested during the Skynet downlink trials. The uplink synchronization portion will be tested in future Skynet uplink trials.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AC	Alternating Current
BCD	Binary Coded Decimal
BW	BandWidth
C25	TMS320C25 Digital Signal Processor
CRC	Communications Research Centre
DC	Direct Current
DIP	Dual Inline Package
DND	Department of National Defence
DREO	Defence Research Establishment Ottawa
DSP	Digital Signal Processor
EHF	Extremely High Frequency
EPLD	Erasable Programmable Logic Device
EPROM	Erasable Programmable Read Only Memory
FSK	Frequency-Shift-Keying
GT	Ground Terminal
h	Suffix indicating a hexadecimal number
hex	Hexadecimal
HSC	Hopping Synthesizer Controller
IF	Intermediate Frequency
I/O	Input/Output
ISA	Industry Standard Architecture
LED	Light Emitting Diode
LO	Local Oscillator
LSB	Least Significant Bit
MILSATCOM	Military Satellite Communications
MSB	Most Significant Bit
PC	Personal Computer
PLCC	Plastic Leadless Chip Carrier
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read Only Memory
SATCOM	Satellite Communications
sync	Synchronization
synth	Synthesizer
TTL	Transistor Transistor Logic
XF	eXternal Flag

1. Introduction

1.1 The Problem

Defence Research Establishment Ottawa is pursuing an in-house research activity in spread-spectrum technology to support development of robust, anti-jam satellite communications for the military. The MILSATCOM (Military Satellite Communications) group is developing a system simulator to research some of the critical techniques such as spread-spectrum synchronization. Any work in this area requires a frequency hopping synthesizer, for each end of the link, that is under control of the Synchronization Controller. The Comstron FS2000 synthesizer takes a frequecy word from an external port and switches to a corresponding frequency within 1 μs. This report describes a Hopping Synthesizer Controller (HSC) which generates the frequency word at the appropriate time and responds to the commands of the Synchronization Controller. The Synchronization Controller is the processor that supervises all aspects of synchronization including adjusting internal clocks and directing the Hopping Synthesizer Controller.

1.2 Hopping Synthesizer Controller Within a System

Fig. 1 shows a simplified ground terminal block diagram using two Hopping Synthesizer Controllers. The transmit user data is collected by the Frame Multiplexer and passed to the HSC as necessary. The transmit HSC drives the Hopping Synthesizer to produce the modulated and hopped signal (including Doppler precorrection) which is then upconverted and transmitted to the satellite. The signal received from the satellite downlink is downconverted, with the downlink HSC causing the Hopping Synthesizer to dehop the signal as well as to postcorrect for Doppler offsets. The signal is then demodulated and demultiplexed to the various users. In both the uplink and downlink chains, the Synchronization Controller commands the HSCs. The Synchronization Controller also provides the Doppler information obtained from the Ephemeris Processor. A payload diagram would be similar, but there would be no requirement for synchronization or ephemeris.

1.3 Hopping Synthesizer Controller Implementation

The original design for a Hopping Synthesizer Controller did not involve a microprocessor. The first design was found to be too inflexible and did not properly support the Synchronization Controller. Based on the information learned, the current design included a digital signal processor to perform the calculations and respond to external commands. This is the implementation described in this report. The calculations involve producing a random hop frequency, adding frequency-shift-keying (FSK) modulation and channel selection, precorrecting the Doppler offsets, and formatting the frequency word for the synthesizer. The Synchronization Controller uses a scheme which requires rapid changing of the hop frequency during the detection of synchronization hops. The HSC supports the Synchronization Controller by precomputing the synchronization hop frequencies and rapidly switching them on demand. The Synchronization Controller can override any of the default parameters of the frequency calculation. The HSC is configurable for use in uplink or downlink mode and for either ground terminal or satellite operation. The HSC can also be configured to control the Comstron or Sciteq synthesizers.
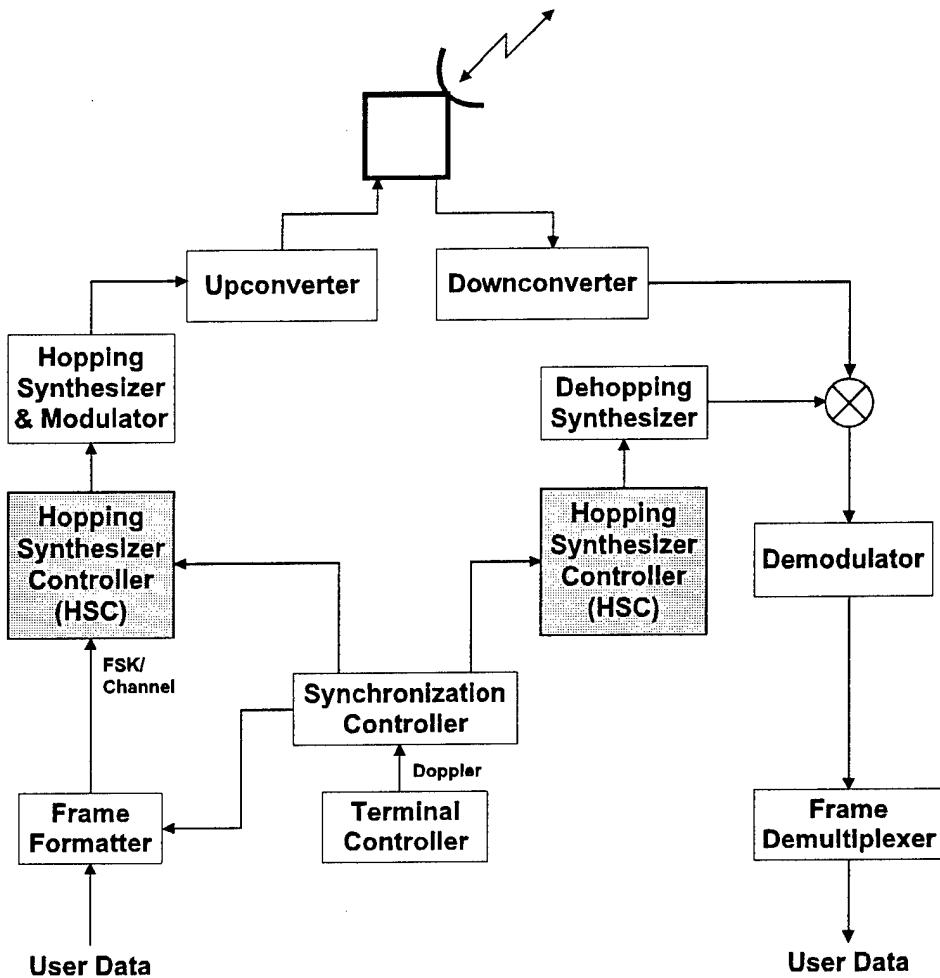
Fig. 1. Ground terminal simplified block diagram.

1.4   Synchronization Background

Synchronization is necessary in a spread spectrum system to ensure that the ground terminal and payload clocks are lined up. Otherwise, communications would be impossible because the two would be at different parts of the hopping pattern. Downlink synchronization occurs when the ground terminal receives the downlink synchronization hops sent by the payload. Based on measurements of these hops within the ground terminal, the ground terminal's downlink clock is adjusted to match that of the payload.

Because the delay to a satellite can change and is not always known accurately, uplink synchronization is also required. In this case, the ground terminal sends an uplink synchronization probe to be processed by the payload. The payload measures the timing error and sends back an uplink synchronization response to the ground terminal. This response is used by the ground terminal to adjust its uplink clock.

Downlink synchronization must always be done first so that the ground terminal can properly decode the uplink synchronization responses. Downlink synchronization starts with coarse acquisition of the downlink synchronization hops, followed by fine acquisition and verification. Once downlink

synchronization has been achieved, it is maintained by downlink tracking. Uplink synchronization starts with probing, then verification and is maintained using uplink tracking.

1.5   Report Outline

The reports consists of two major sections, software and hardware, as well as several appendices containing detailed technical information. The software section covers the structure of the digital signal processor software, the command processing sequence for Synchronization Controller commands, and the algorithms used. These algorithms cover the frequency calculations, binary to BCD conversion to format the word for the synthesizer, and the random number generation for the hop frequency pattern.

The hardware section covers the HSC description, the details of the digital signal processor, and the erasable programmable logic device (EPLD) used for interfacing and the steps taken to alleviate a timing problem caused by propagation delays in the EPLD.

The appendices include the user's guide, command reference, external interfaces, hardware details, firmware/software details, and PC (Personal Computer) test program documentation.

2. Software

2.1 General

This section describes the software that is programmed into an erasable programmable read-only memory (EPROM) for the HSC, sometimes referred to as firmware. The EPLD is programmed as well, but since it replaces interface logic chips, the EPLD details can be found in the hardware section. All software listings and associated command files are given in Appendix E.

The HSC operates in one of four modes: STOP mode where hopping is stopped, RUN mode where normal hopping takes place, and two synchronization modes. In the synchronization modes (DLSYNC and ULSYNC), precomputed hops are invoked by the Synchronization Controller.

The most commonly used commands are the LOAD commands to change the calculation parameters and the mode commands to switch between modes. All calculation parameters can be restored to their default settings by the DEFAULTS command. The DLGO and ULGO commands are used in DLSYNC and ULSYNC modes respectively to change frequencies rapidly. The remaining commands are used for debugging: READ to allow the Synchronization Controller to examine the HSC's memory, and CHANGE commands to control the latch timing of the hopping synthesizer.

In the following sections the software structure, Synchronization Controller command processing and the calculation algorithms are presented.

2.2 Program Structure

The software consists of three files: MAIN, CALC and RAN. The MAIN file includes the main program and subroutines for command processing (INSTR), setting default values (SETDEF) and miscellaneous short support routines. CALC contains the calculation subroutines for frequency computations (CALC), the binary to BCD conversion (BCD32 and BCDINI) and a 32 x 32 to 64-bit signed integer multiply (MUL32). The random number generation routines can be found in RAN including the random number routine (RAN and RANINI) and a routine to step ahead in time (RANJMP). The detailed algorithms (frequency calculations, BCD conversion, and random number generation) can be found in Section 2.4.

The HSC operates in one of several modes. The various modes used in the HSC are shown in Fig. 2. On power-up the HSC is in STOP mode. This mode is used for initialization by using any of the LOAD commands, DEFAULT command, CHANGE commands or READ command. Once the HSC is initialized, then one of DLSYNC, ULSYNC or RUN commands should be given depending on the application.

The four mode commands (STOP, RUN, DLSYNC, ULSYNC) can be easily distinguished because the STOP command has a value of 0, the RUN command is negative (value 8000h = -32768) and the two sync (DLSYNC and ULSYNC) are positive (value 4000h = 16384 and 5000h = 20480).

The DLSYNC mode is entered by the ground terminal HSC to perform downlink synchronization. In this mode, the downlink synchronization hops are precomputed and the synthesizer is switched rapidly on the DLGO commands. Once coarse synchronization has been achieved, the RUN command is used to continue with normal hopping.

The ULSYNC mode is entered by the ground terminal HSC to perform uplink synchronization. Once this mode is entered, the FCALC command should be used to precompute one frames worth of synchronization probes. The synthesizer can then be switched rapidly on the ULGO commands. Once coarse uplink synchronization has been achieved, one should switch to RUN mode for fine uplink synchronization and normal hopping.

4
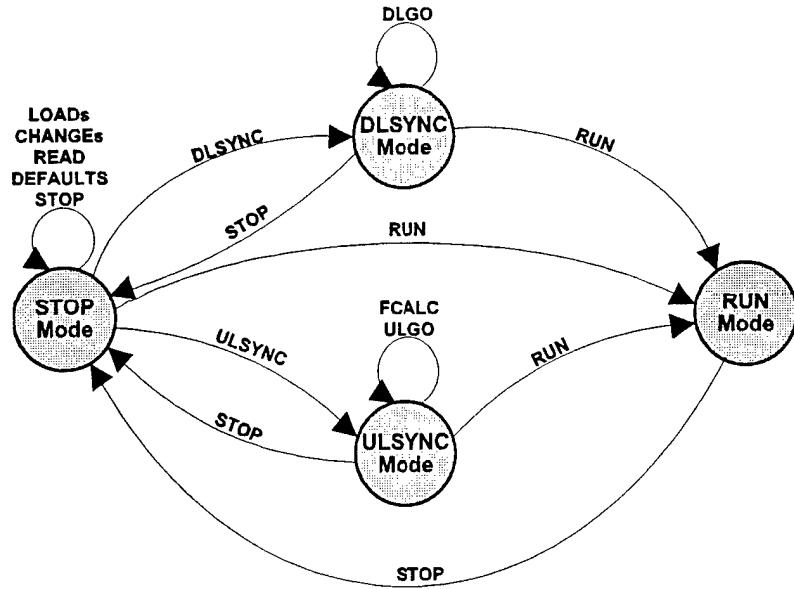
Fig. 2. Hopping Synthesizer Controller modes of operation.

The RUN mode is entered immediately after initialization for an HSC in the payload simulator or after coarse synchronization for an HSC in the ground terminal. RUN mode, where normal hopping occurs, is the steady-state mode of the HSC.

2.3 Command Processing

The effectiveness of the HSC hinges on rapid response in DLSYNC and ULSYNC modes. This is because of the synchronization schemes that the HSC was designed to support.

2.3.1 Downlink Synchronization

The downlink synchronization scheme involves the detection of a burst of four synchronization (sync) hops with predetermined modulation and data. Each sync hop is separated by a blank hop during which the detection computations for the previous hop occur. If a sync hop is detected, the next hops in the burst are used for confirmation. Once the detection of a burst of four sync hops is detected, another confirmation involving all four sync hops of the next burst is required. The first hop of the second burst is precomputed whereas for the final three hops the HSC is switched into normal RUN mode. A false alarm is deemed to have occurred if the initial sync hop is detected and the subsequent sync hops are not detected.

In DLSYNC mode, the HSC is able to respond to commands and switch frequencies in 3 μs. To allow such a rapid response it is necessary that the command decoding be very simple. This is accomplished by reducing the number of different commands to a minimum and by choosing opcodes that can be easily decoded.

Within DLSYNC mode, the DLGO commands request the rapid change of frequency. There are five possible DLGO commands corresponding to the five precomputed frequencies. To simplify command decoding, each DLGO command opcode is identical to the address of the precomputed frequency. The DLGO command is loaded into an index register and then the BCD frequency that it points to is sent to the synthesizer.

5

To summarize DLSYNC mode command processing, first the processor tests for STOP (zero flag set means switch to STOP mode), then tests for RUN (sign bit set implies switch to RUN mode) and otherwise uses the opcode as the address of the frequency word to send to the synthesizer. There is no valid opcode or range checking, so if an illegal opcode (including other commands, non-commands and DLSYNC) is received then indeterminate results could occur. Response time is more important than robustness (with respect to illegal commands) while in DLSYNC mode.

2.3.2   Uplink Synchronization

For uplink synchronization, two bursts of 16 probes each are generated per frame. Each probe is sent under control of the ground terminal Synchronization Controller using an ULGO command. A response corresponding to the timing error of the transmitted probes is subsequently sent from the payload simulator to the ground terminal simulator to allow the ground terminal to make appropriate adjustments to its clock. All of the probes for a frame are computed ahead of time using the FCALC command. For unused frames, the FCALC command is still required to ensure the HSC clock is advanced. Once the ground terminal's clock is coarsely adjusted, the HSC is switched into RUN mode to continue on with fine synchronization and tracking.

The opcodes used for uplink synchronization (ULGO, FCALC) are similar to the downlink case in that they are chosen to speed command processing and hence frequency changes.

2.3.3   Loads

The LOAD commands are similar to the DLGO and ULGO commands in that a particular LOAD command opcode is also the address to be used. They can be distinguished from the DLGO and ULGO commands because they use different areas of memory (DLGO commands start at 60h, ULGO commands start at 200h and LOAD commands start at 300h). LOAD commands are followed by the low 16-bit word and then the high 16-bit word of the 32-bit value to be loaded.

2.3.4   Other Commands

The READ command was added strictly for debugging. The lower 12 bits of the READ command specify the memory location to be read. The READ command is of the form 2*mmm*h where *mmm* is the address of the memory location to be read. The remaining commands, DEFAULTS, CHANGE HOP, CHANGE IMMEDIATE, were assigned distinct values of 0400h, 0500h and 0501h respectively.

2.3.5   Processing in Various Modes

In STOP mode, the processor waits only for commands and processes them once all the words of the command have been received. For example, all three words of a LOAD command are buffered before the command is processed. In STOP mode only legal commands are processed and if illegal opcodes are received they are ignored.

While in RUN mode, the commands are processed only once per hop. After the new frequency is computed and if all the words of the command are received then one command will be processed. After that, the processor waits for the next hop. Invalid opcodes are ignored when received in RUN mode.

6

2.4   Algorithms

There are three main algorithms used within the HSC. The first and most important one is the Frequency Computation algorithm which determines how the random number, modulation, channel number and configuration affect the desired frequency. The second algorithm formats this frequency and controls the RF (radio frequency) output for the appropriate synthesizer. The last algorithm is a pseudo-random number generator used to determine the hop location within the hopping bandwidth.

2.4.1   Frequency Computations

All frequency computations are performed in the routine CALC according to the equations given below. The different HSC configuration options (uplink or downlink, ground terminal or payload) are handled by different values of the arguments. An example of these values can be found in Table 1 at the end of this section. Fig. 3. shows graphically the various frequencies used.
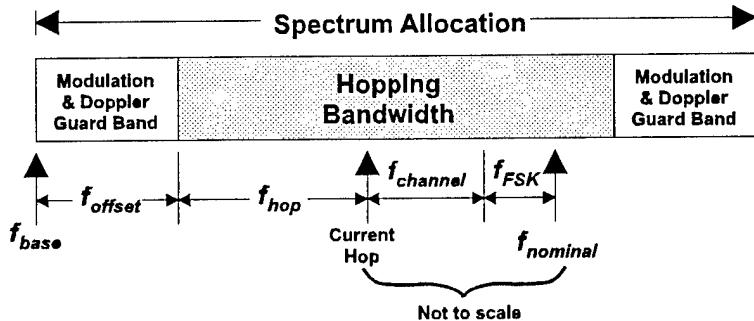
Fig. 3. Graphical depiction of frequencies used.

In the calculations shown below, certain values such as the bandwidth (BW) scale factor BW_{SCALE} and the base frequency f_{base} change depending on the HSC configuration selected. For illustrative purposes, the values used will be those associated with the uplink from a ground terminal. This is the most complex case since it includes modulation. All the other cases are subsets of this case.

First, the available hopping bandwidth will be calculated to produce the scale value for the random number that provides the random hop frequency. Then the frequency offset associated with modulation and channel selection will be computed. These will be summed with the fixed frequency offsets at which point Doppler is added. Finally the value needed for the synthesizer will be derived.

The available hopping bandwidth is 2 GHz. If a 500 kHz guard band for Doppler is left at both the lower and upper edges, there is only 1.999 GHz remaining. A multichannel system could have a bandwidth as large as 1 MHz, giving a useable hopping bandwidth of 1.998 GHz.

The placement of the hop within the hopping bandwidth is determined by a pseudorandom hop number (N_{hop}) that is generated for each hop. This provides a random number from 0 to 16777215 which is multiplied by BW_{SCALE} (9990000 = 1.1909008026 x 2^{23}) and then scaled by 2^{-23}. The resultant, f_{hop}, is the frequency offset of the hop from the lower edge of the hopping bandwidth. f_{hop} is in units of 100 Hz and given by

$$f_{hop} = \frac{N_{hop} \times BW_{SCALE}}{2^{23}} \qquad (1)$$

7

The frequency offset associated with FSK modulation and channel spacing, f_{FSK} and $f_{channel}$, are derived from the 8-bit FSK/Channel input. The lower 3 bits, N_{FSK}, select one of the 8-ary tones. The channel selection, $N_{channel}$, is from the upper 5 bits to select one of 32 channels. The channels are 8 tones wide with no interchannel spacing. The intertone spacing, f_{space}, is 360, which (in units of 100 Hz) corresponds to a spacing of 36 kHz. f_{FSK} and $f_{channel}$ are given by

$$f_{FSK} = f_{space} \, N_{FSK}$$

$$f_{channel} = 8 f_{space} N_{channel}$$

(2)

Neglecting, for the moment, Doppler frequency offset, the nominal frequency, $f_{nominal}$, is the sum of the low edge of the hopping bandwidth f_{base}, the lower edge guard offset to allow for negative Doppler f_{offset}, the hop frequency, channel selection and modulation. Thus,

$$f_{nominal} = f_{base} + f_{offset} + f_{hop} + f_{channel} + f_{FSK}$$

(3)

Doppler effects between a moving satellite and ground terminal are precorrected by the ground terminal prior to transmission of the uplink and postcorrected after reception of the downlink. The satellite does not make any modifications for Doppler. The actual transmitted frequency must be precorrected for Doppler frequency offset (or to correct the Doppler shifted received frequency) so it appears as $f_{nominal}$ to the satellite. The transmit (or receive) frequency, f_{rx_tx}, is the nominal transmit frequency plus the Doppler frequency. This Doppler is the product of the nominal frequency and the Doppler factor γ. A positive Doppler factor is used here when the satellite is moving away from the ground terminal thus compensating for a negative Doppler shift. On the receive side of the ground terminal, the Doppler factor sign would be reversed for postcorrection of Doppler shift. For calculation accuracy with fixed point arithmetic, N_γ is used (equal to γ times 2^{32}). f_{rx_tx} is given by

$$f_{rx\_tx} = f_{nominal} + f_{nominal} \cdot \frac{N_\gamma}{2^{32}}$$

(4)

A synthesizer has a limited frequency range and is usually part of an up or downconversion chain. The frequency desired from the hopping synthesizer, f_{synth}, is the transmitted (or received) frequency less the up or downconverter local oscillator (LO) frequency f_{LO} as shown in Fig. 4.

$$f_{synth} = f_{rx\_tx} - f_{LO}$$

(5)

As an example, the default values corresponding to an f_{synth} of 645-695 MHz for the key elements of the ground terminal (GT) uplink are given in the Table 1 below:

Table 1. Sample calculation parameters for the Comstron synthesizer.

BW_{SCALE}	f_{space}	f_{base}	f_{offset}	N_γ	f_{LO_Com}
250000 (50 MHz)	360 (36 kHz)	445950000 (44.595 GHz)	0 (No guard offset)	54 (561 Hz Doppler on 44.595 GHz)	439500000 (43.95 GHz)

8
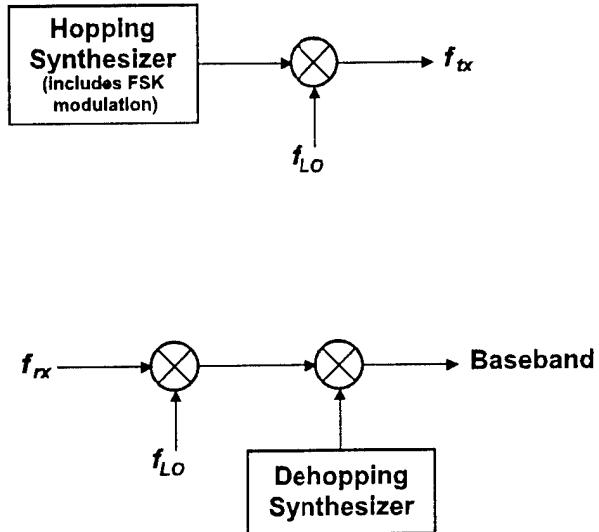
Fig. 4. Upconversion/downconversion diagram.

2.4.2 Comstron BCD Formatting

The resultant frequency, f_{synth}, is the desired binary frequency for the synthesizer in units of 100 Hz. Unfortunately, the Comstron synthesizer requires a BCD (Binary Coded Decimal) command frequency. The conversion from binary to a 32-bit BCD word is accomplished in the routine BCD32. Different synthesizers, such as the Sciteq synthesizer, use different formats and require different formatting routines. This is described in Section 2.4.3.

The binary to BCD conversion algorithm involves repeated binary subtractions similar to binary long division. Long division uses the same dividend (though shifted) whereas the BCD conversion uses a different dividend for each decade. This dividend is 8 times the decade of conversion and is shifted to the most significant position (it would be $8 \times 10 \times 16^M$ for the tens decade where M depends on the size of the conversion). The subtractions are in fact conditional subtractions - if the subtraction leaves a positive result then it is performed, producing a 1 for the conversion, otherwise there is no subtraction giving a 0. The TMS320C25 (C25) has a conditional subtraction instruction, but it only works with a 16-bit dividend. A 32-bit algorithm equivalent to this instruction is used for the high 16 bits, and then the conditional subtraction is used for the low 16 bits.

9

The binary number to be converted is given by **binary** and the BCD result is given by **BCD**. If there are **N** decades to be converted, the algorithm is:

```
BCD = 0
do i = 0  to  N-1
    dividend = 8 x 10^(N-i-1) x 16^i
    do j = 0  to  3
                if dividend ≤ binary then
                    binary = binary - dividend
                    BCD = BCD x 2 + 1
                else
                    BCD = BCD x 2 + 0
                endif
                binary = binary x 2
    enddo
enddo
```

Table 2 shows an example of the formatting algorithm using a value to be converted (**binary**) of 279 or 177h. It will be a 3 (**N**) decade conversion. The first dividend is computed as $8 \times 10^2 \times 16^0$ = 800 = 320h. **BCD** is initialized to 0.

The table shows the values of the indexes: i is used to count the decades and j is used to count the bits within each decade. The operation is shown for each step indicating the results of the test and the bits that is added to **BCD**. The value to be converted (**binary**) changes at each step as does the answer (**BCD**). The dividend changes every decade.

As can be seen from the table, the conversion is complete after 3 decades giving the BCD output required. The hexadecimal value of the answer is the BCD form of the original value.

The doubling operations of the algorithm are actually implemented using a left shift on the processor. The conditional subtraction instruction on the C25 has the additional feature that the answer is shifted into the low bit of the value. Thus only one 32-bit accumulator is needed and both shift operations can be done with one shift of the accumulator. The accumulator starts with the input value and ends up with the BCD output value.

To speed up the conversion process, a table of dividends is loaded into random access memory (RAM) by the routine BCDINI. This routine must be invoked once at the beginning of the main program, prior to any calls to the conversion routine BCD32.

Table 2. Comstron BCD Formatting Example.

i	j	Operation	Binary	Dividend	BCD
-	-	Initial Values	279 = 117h	800 = 320h	0
0	0	Binary < Dividend No subtraction Double BCD and add 0 Double binary	279 = 117h → 558 = 22Eh	800 = 320h	0
	1	Binary < Dividend No subtraction Double BCD and add 0 Double binary	558 = 22Eh → 1116 = 45Ch	800 = 320h	0
	2	Binary ≥ Dividend Subtract Double BCD and add 1 Double binary	1116 = 45Ch → 632 = 278h	800 = 320h	1
	3	Binary < Dividend No subtraction Double BCD and add 0 Double binary	632 = 278h → 1264 = 4F0h	800 = 320h	2
1	0	Binary < Dividend: 0	1264 = 4F0h → 2528 = 9E0h	1280 = 500h	4
	1	Binary ≥ Dividend: 1	2528 = 9E0h → 2496 = 9C0h	1280 = 500h	9
	2	Binary ≥ Dividend: 1	2496 = 9C0h → 2432 = 980h	1280 = 500h	19 = 13h
	3	Binary ≥ Dividend: 1	2432 = 980h → 2304 = 900h	1280 = 500h	39 = 27h
2	0	Binary ≥ Dividend: 1	2304 = 900h → 512 = 200h	2048 = 800h	79 = 4Fh
	1	Binary < Dividend: 0	512 = 200h → 1024 = 400h	2048 = 800h	158 = 9Eh
	2	Binary < Dividend: 0	1024 = 400h → 2048 = 800h	2048 = 800h	316 = 13Ch
	3	Binary ≥ Dividend: 1	2048 = 800h → 0	2048 = 800h	633 = 279H

2.4.3   Sciteq Formatting

The Sciteq synthesizer frequency is controlled by 35 bits in a split-binary format. There are 6 bits of coarse frequency control where the least significant bit (LSB) is 25 MHz, the most significant bit (MSB) is 800 MHz and the control range is 0 to 975 MHz. The fine frequency control is 29 bits wide where the LSB is 65.1925802231 mHz and the MSB is 17.5 MHz. The fine control range is 5 MHz to 29.99999999 MHz. The entire output is offset by 1 GHz. The equation describing the output frequency is given below:

11

$$f_{Sciteq} = 995 + 25\,N_{coarse} + \frac{17.5}{2^{28}}\,N_{fine} \quad \text{[MHz]}$$

where: f_{Sciteq} is the frequency output of the Sciteq synthesizer   **(6)**
N_{coarse} is the coarse frequency word (6 bits)
N_{fine} is the fine frequency word (29 bits)

Since all calculations on the HSC are performed to a resolution of 100 Hz, it is not necessary to drive all of the bits for the fine frequency. Only 32 bits are available from the latches and the fine word plus the coarse word require a total of 35 bits. (The 9th bit of the fine frequency word which is the 33.38 Hz bit, was the limit chosen to minimize the number of fine bits while maintaining a minimum accuracy of the calculations.) All fine bits less than 33.38 Hz will be forced to 0. This reduces the fine frequency word to 20 bits (and the total to 26 bits):

$$f_{Sciteq} = 995 + 25\,N_{coarse} + \frac{17.5}{2^{19}}\,N_{fine\_HSC} \quad \text{[MHz]}$$

where: f_{Sciteq} is the frequency output of the Sciteq synthesizer   **(7)**
N_{coarse} is the coarse frequency word (6 bits)
N_{fine_HSC} is the HSC driven fine frequency word (20 bits)

For a desired synthesizer frequency of f_{Sciteq}, the coarse and fine words can be computed with:

$$N_{coarse} = \left\lfloor \frac{f_{Sciteq} - 1\ \text{GHz}}{25\ \text{MHz}} \right\rfloor$$

    (8)

$$N_{fine\_HSC} = \left\lfloor \left(\frac{2^{19}}{17.5\ \text{MHz}}\right)(f_{Sciteq} - N_{coarse} \cdot 25\ \text{MHz} - 995\ \text{MHz}) \right\rfloor$$

The resulting composite frequency word to be sent to the latches for the frequency synthesizer is:

$$N_{latch} = 2^{20} N_{coarse} + N_{fine\_HSC}$$

    (9)

This corresponds to the following Sciteq synthesizer word:

$$N_{Sciteq} = 2^{9} N_{latch}$$
$$= 2^{29} N_{coarse} + 2^{9} N_{fine\_HSC}$$

    (10)

12

For example, assume that the desired frequency for the Sciteq, f_{Sciteq}, is 1.5432 GHz. Using (8), N_{coarse} is 21 or 15h and N_{fine_HSC} is 695056 or A9B10h. The value to be sent to the latch, using (9), is 22715152 or 015A9B10h. This value in the latches causes the Sciteq to be driven with 11630157824 or 2B5362000h. The resultant frequency output will be 1.54319999695 GHz (only a difference of 3 Hz from the desired frequency).

2.4.4 Controlling Synthesizer Output

It is convenient to have the ability to disable the RF output on a hop basis to produce a burst signal. On the Sciteq, there is an output control line in the interface that allows disabling of the radio frequency (RF) output by the HSC. The Comstron synthesizer does not have this feature so the effect was emulated by inverting the 1 GHz bit (for example, a 1.5 GHz signal would be sent at 0.5 GHz and a 0.8 GHz signal would be sent at 1.8 GHz).

The control of the RF output originates from the FSK/Channel interface which is sampled one hop early on the hop rising edge. For the Sciteq, the output is switched off by setting the RFoff control line on the synthesizer interface. For the Comstron, the frequency is adjusted by an exclusive-or on the 1 GHz bit to cause it to be inverted.

The final result of disabling the RF output is either a 30 dB attenuation of the output for the Sciteq synthesizer or an out of band hop for the Comstron synthesizer.

2.4.5 Random Number Generation

In an operational system, the hop frequency will be chosen using an encrypted sequence. For test purposes, it is impractical to acquire and use an encryption device so a simple random number generator was chosen. A simple implementation of a 31-bit multiplicative linear congruential random number generator is given in [1]. The basic equation is given below where s_{n+1} is the new random number (and next seed), s_n is the current seed and a is the multiplier. (For the hop sequence the value of $a = 16807$ was chosen because it is known to produce a long sequence). The random number is

$$s_{n+1} = a s_n \mod(2^{31} - 1) \tag{11}$$

The product $a s_n$ will be between 32 and 62 bits long. This product can be split into two numbers: q for the lower 31 bits and p for the upper bits then the two components are 31 bits or less and given by

$$a s_n = 2^{31} p + q \tag{12}$$

Depending whether the sum $(p + q)$ overflows or not, (11) can be expressed as (note that the modulo operation has been avoided)

$$s_{n+1} = \begin{cases} p + q & ; p + q < 2^{31} \\ p + q - 2^{31} + 1 & ; p + q \geq 2^{31} \end{cases} \tag{13}$$

This can be thought of as discarding the overflow and incrementing the result when an overflow occurs. Equivalently, adding the overflow to the result gives

13

$$s_{n+1} = p + q + \text{overflow} \qquad\qquad \textbf{(14)}$$

Equations (12) and (14) result in compact code for 31-bit random number generation. The routine RAN computes this number using these equations. The linear congruential random number generators cannot use a seed of zero, so prior to any random number generation, RANINI should be called to initialize the seed to 1.

Occasionally, it is necessary to set the clock ahead in time. The routine RANJMP rapidly steps through the random numbers until the one appropriate for the specified time-of-day is computed. There is no known means of changing time rapidly with these generators, so this brute force method must be employed.

The synchronization testing requires that the sequence of random numbers be of some finite length (so one does not have to wait forever in the case of an error in the synchronization methods). For simplicity, a length less than 2^{16} was chosen giving a sequence length of 61440. It was important that this length be an integral number of sync hop burst periods. The random number generator seed is automatically reset to 1 at the end of the sequence to begin the whole process again.

3.   Hardware

3.1   General

The block diagram of the controller is shown in Fig. 5. Within the HSC, the digital signal processor (DSP) is used to perform the computations and respond to commands. Internal RAM is used for variables and external EPROM is used for program memory. There is also a latch which is used to hold the frequency computed by the DSP for the synthesizer. Finally, there is a custom interface that latches Synchronization Controller commands, performs handshaking, generates control signals and provides the status of the configuration DIP (dual in-line package) switches.
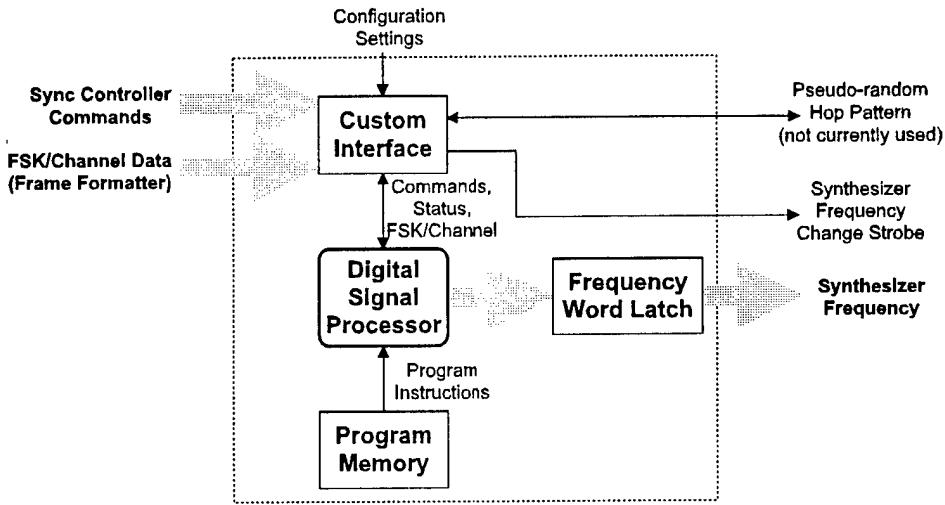
Fig. 5. Hopping Synthesizer Controller block diagram.

3.2   Board Construction

The HSC is driven by a TMS320C25 signal processor that performs the computations and responds to commands from the Synchronization Controller. The C25 program is stored in 2 x 7C261 8k-byte fast EPROMS. To buffer and latch the outputs to the frequency synthesizer, 4 x 74LS374 octal latches are used. 5 x 74LS244 are used to buffer offboard signal and data lines. The EPM5128 EPLD provides all the interface logic: hop clock generation from the 40 MHz clock, configuration control, Synchronization Controller interface (latch and handshaking), C25 interface (port select, status and data ports), synthesizer strobe and onboard latch strobes. Finally, a 74LS221 dual monostable multivibrator is used for power-up reset and a crystal oscillator is used for the 40 MHz clock. This circuit was implemented on a custom printed circuit board. The HSC schematic is given in Fig. D1 and D2 of Appendix D.

3.3   TMS320C25 Digital Signal Processor

The C25 is the processor used in the HSC. This digital signal processor has a good architecture for the frequency computations and is sufficiently flexible to perform the other duties. The component used was a 40 MHz part in a plastic leadless chip carrier (PLCC) package. The C25 has two

modes of operation: microprocessor mode which uses external read-only memory (ROM) and microcomputer mode which uses internal and external ROM. On this board, the C25 was operated in microprocessor mode with program memory located in an external EPROM. There is sufficient on-chip RAM for all data storage so there is no need for external RAM. This microprocessor has separate program and data storage capacity of 64k 16-bit words each. The C25 also has an input/output (I/O) space with 16 read/write ports addressed by 4 bits. On this board, only the lower two of four address bits are decoded for four different I/O ports (resulting in 3 extra shadow groups). Of the four decoded ports, only two are used for specific functions depending on whether the port is being read from or written to. The remaining two are unused and may be used for future expansion. The associated memory and I/O maps are given in Tables 3, 4 and 5.

Table 3. Program memory map.

Program Address	Program Memory Use
0000-1FFF	Program EPROM
2000-3FFF	Shadow* of EPROM
4000-5FFF	Shadow of EPROM
6000-7FFF	Shadow of EPROM
8000-FFFF	Shadows of EPROM

* a shadow occurs when the upper address bits are not decoded so accesses to all memory addresses that match the lower decoded bits, access the same location.

Table 4. Data memory map.

Data Address	Data Memory Use
0000-0005	On-Chip Registers
0006-005F	Reserved
0060-007F	On-Chip RAM
0080-01FF	Reserved
0200-03FF	On-Chip RAM
0400-FFFF	External (unused)

Table 5. I/O port map.

Port	Read	Write
0	Command Register	Synthesizer Latch Low
1	Status Register	Synthesizer Latch High
2	Unused	Unused
3	Unused	Unused

3.4 Erasable Programmable Logic Device

EPLDs are integrated circuits that consist of many macrocells. Each macrocell is a configurable flipflop fed by an AND/OR matrix. Most macrocells are connected to an output pin allowing combinatorial or registered output. The larger EPLDs have embedded macrocells that have no output (for internal latches or shift registers) and expander terms (for more complex expressions). The combinatorial equations, flipflop configuration and whether a pin is input, output or both is programmed into the EPLD. Modifications can be made to the EPLD by erasing (using an ultraviolet light) and then reprogramming the EPLD. The EPLD used in this design is the Altera EPM5128 which consists of 128 macrocells in a 68 pin PLCC package. The following paragraphs describe the circuit within the EPLD and refer to the EPLD schematic in Fig. E1 of Appendix E.

3.4.1 Clock Generation and Status

The clock divider section (in the extreme upper left corner) provides the clocks required on chip: 40 MHz, 20 MHz and the internal hop clock. Below this section is the upper byte of the status register (the lower byte is the FSK/Channel information) which reflects the current configuration switch positions (R8-R11), the status bits for 1 GHz Offset (R12), Synchronization Controller command ready (R13), and the hop clock (R15). R14 always reads 0. The hop clock is chosen from the internal clock divider or from the Synchronization Controller interface external clock depending on the state of the Int/Ext configuration switch. The eight wide multiplexer to the right chooses between the upper byte of the status register and the upper byte of the latched Synchronization Controller command word. The high byte of status or the entire word of the Synchronization Controller command is then passed through a tristate buffer to connect to the C25 data bus.

3.4.2 Command Latch

The Synchronization Controller command word previously mentioned is latched on the rising edge of the SStrobe line. The rising edge also causes a flip-flop to be set indicating that a command is ready to be read by the C25. This flip-flop is automatically cleared when the command is read. The value of the flip-flop is reflected in bit 13 of the status register as well as on the BIO pin of the C25. The latter is used to exploit the fastest possible bit test on the C25 for rapid response during DLSYNC mode.

3.4.3 Address Decoder

The lower left corner of the EPLD schematic covers the decoding of the various addresses. The port assignment can be found in Table 5 provided previously. The unused ports are available for

17

future expansion. Note that only the lower two bits of the four-bit port address are decoded.

The data for a read from port 1 is provided by two sources. The upper 8 bits are provided by the status register within the custom interface EPLD. The lower 8 bits come from the FSK/Channel interface whose tristate buffer is enabled by the Rd_FSK line at the same time. The purpose of including the Clkout1 pulse for the decoder is to provide an early rising edge that, when delayed by the propagation delay of the EPLD, will rise while the data is still valid during the write cycle. This edge is used by the synthesizer latches. Further details on this reasoning and how it affects modifications to the design are presented in Section 3.5. To further align the synthesizer latches, both strobes are delayed by 50-100 ns through a pair of 2-bit shift registers clocked at 20 MHz.

3.4.4 Synthesizer Strobe

In the centre of the schematic is the logic necessary for the strobe which causes the synthesizer to change frequency (as opposed to the strobe for the on-board synthesizer latches that hold the frequency to the synthesizer). There are two possible strobe timings selected by the XF pin (external flag output, a general purpose output pin that is used here to select the strobe) of the C25: immediate and hop-edge. The immediate strobe is used during the DLSYNC and ULSYNC modes and causes the synthesizer to change frequencies as soon as a value is written to the synthesizer high latch. The hop-edge strobe is used in normal RUN mode and causes the frequency to change at the rising edge of the hop.

The rising edge of the strobe (of the type selected) enables the shift register to clock out a pulse of the duration and shape necessary for the synthesizer. The enabling line is reset when the pulse has been shifted out.

3.5 Timing Consideration

A problem occurred when using the EPLD to strobe the onboard latches. The C25 holds the data for 30 ns (minimum) before and for 15 ns (minimum) after the write strobe. Unfortunately, the propagation delay of 30 ns in the EPLD means the strobe to the latch would be up to 15 ns late. Therefore, instead of using the NStrb pulse from the C25, the Clkout1 pulse (gated by the write line) was used because this combination precedes the strobe pulse by 25 ns. When this is delayed by 30 ns it produces a rising edge within the period for which the data lines are valid.

The onboard latch strobe portion of the design is heavily dependent on the speed and propagation times of the C25 and the EPLD. As the EPLD speed is increased, there will be less and less setup time for the latches. If the C25 were to speed up (faster clock speed or lower propagation times), the design would not work. If either the C25 or the EPLD were replaced by chips with longer propagation times, the synthesizer latch strobes would have to be redesigned because the Clkout1 pulse is no longer in the right place.

4. Testing and Modifications

A description of the testing of the initial implementation of the HSC is given in the following section. The HSC has also been updated and tested to support the uplink synchronization experiments. The strobe glitch problem that was identified during the test of the modified HSC is presented along with the corrective measures taken. In addition, several modifications made to the HSC to improve its functionality are outlined.

4.1 Testing

Testing of the HSC occurred in three stages: post-construction testing, EHF transmitter/receiver testing and Skynet downlink synchronization trials.

Upon initial completion of the HSC, performance testing was done. It was determined that the processor was idle for about one-half of the time. At maximum speed, the HSC will support hop rates as high as 25000 hops/s.

Upon construction of the 44/20 GHz transmitter and receiver at DREO and Communications Research Centre (CRC), the hopping was tested using both the Comstron and Sciteq synthesizers driven by the HSC. During these tests, the HSC worked, but an error rate of 2 in 10^6 was observed even in the absence of injected noise. This could have been due to the Comstron synthesizers (which have demonstrated problems throughout the last six years), the long cable used during the tests, or the HSC glitch problem (which is covered in the next section).

The next stage of testing occurred during the downlink synchronization trials documented in [2]. There were problems observed during downlink trials over two years including a non-zero error rate and occasional synchronization losses. Some, but not all problems, were directly attributed to problems with the Comstron synthesizer. The Comstron synthesizers were sent for repairs several times which improved their performance but some problems persisted. At this time the sensitivity of the Comstron synthesizers to the cable length from the HSC was noted. The cables were then made as short as possible.

At the end of the downlink trials, synchronization and data transfer were achieved. Consequently, successful operation of the HSC was demonstrated.

4.2 Strobe Glitch Problem

During the testing of the modifications to support uplink synchronization, frequency changes were observed while single-stepping using TESTHSC (detailed in Appendix F) even though no changes were commanded. After extensive probing, the problem was isolated to be caused by a glitch on the strobe line for the on-board latches. The glitch was very narrow (less than 10 ns wide) and between 1 and 2 volts. This had the effect of occasionally loading erroneous values into the on-board latches. It did not happen all the time because of the narrowness of the pulse (there was not always sufficient setup time for the latch) and because the logic level was not always a clear "1". It was noted that the glitch was bigger on the upper 16 bit strobe than the lower. As well a much smaller glitch was observed on the synthesizer strobe line.

The timing of the glitch was deterministic and upon examination was found to occur simultaneously with read of the status. The glitch was attributed to EPLD ground bounce occurring when most of EPLD data bus lines are driven from tri-state to low for the status read, as documented in [3]. To eliminate the problem would require changes to the design by inserting tri-state buffers between the EPLD and the bus. It was simpler to treat the symptoms rather than modify the printed circuit board.

To minimize the susceptibility of the latches to the glitches, the fast response 74F374 were replaced with the slower 74LS374. To decrease the magnitude of the glitch to less the 0.5 V, 62 pF capacitors were added on the latch strobe lines and the synthesizer strobe line. The two combined to remove any effects of ground bounce in the EPLD on the circuit.

4.3 Modifications

This section describes the modifications made on the HSC since it was first built six years ago. They include modifications to the startup sequence, changing the default settings, the addition of

the LOAD HOP command, changes made for the Sciteq synthesizer and the addition of support for uplink synchronization.

4.3.1   Startup Sequence

Upon power-up, the HSC initially caused the Comstron synthesizer to display a sequence of digits. By verifying this sequence, one could ensure that all of the interface lines were functioning correctly. At the end of the sequence, the frequency was left at a value of 0 Hz (an illegal value for the Comstron). When connected to the Sciteq, this sequence had no useful purpose. As well, the final frequency value of 0 Hz was not particularly useful.

Improvements were made by changing the sequence such that the first value is the version number of the HSC firmware, followed by the sequence, but ending with the frequency synthesizer at midband. The firmware version and sequence are still of no use on the Sciteq (it has no display) but both the Comstron and Sciteq end up at the middle of the hopping range.

4.3.2   Default Settings

The first use of the HSC was for testing of the EHF receivers and transmitters. These were designed to use the full hopping bandwidth in the 44 GHz and 20 GHz frequencies. Later on, the HSCs were used for the Skynet trials [2] which used much narrower bandwidths. Since future use of the HSC is likely over Skynet, the default power-up parameters were customized for the Skynet trials. Usually, the defaults loaded at power-up will be overridden at the beginning of each test by the Synchronization Controller.

Another aspect of the default settings is the power-up operation modes RUN or STOP. Initially, all payload configurations powered-up in RUN mode for stand-alone operation. This was not desired for the Skynet trials, so now all configurations power-up in STOP mode.

4.3.3   Load Hop Number

To support debugging, a command to load the output latches directly (LOAD LATCH) was implemented. This worked well for the Comstron synthesizer because it used BCD formatting. The Sciteq uses a split-binary format that is hard to compute by hand. So a synthesizer independent way of setting frequency was developed. Instead of loading the latches directly, one instead loads the hop number (LOAD HOP). This allows simple setting of low, mid and high edge-of-band by putting in the minimum, centre, and maximum hop numbers. Once the hop number is specified, the appropriate calculation and formatting are invoked and the results are placed in the latches. Details of LOAD HOP and LOAD LATCH may be found in Appendix B.

4.3.4   Sciteq Synthesizer Specific Modifications

The original HSC was developed to support the Comstron synthesizers. Once the HSC was built, there was an interest in using it with the Sciteq synthesizers.

Because the Sciteq synthesizer does not have the hopping range of the Comstron, it was necessary to double the output of the Sciteq for the uplink. Consequently, the HSC frequency calculations had to be modified to divide by two prior to driving the Sciteq output. This was initially hard coded for the HSC in the payload uplink mode but recently has been changed to a flag that is settable externally and does not depend on the mode.

Another feature of the Sciteq synthesizer is the ability to disable the output using the Reset line. This is useful for burst transmissions and thus, the ability to control the output was added to the

20

HSC. The Sciteq line is controlled using the RFoff line in the FSK/Channel interface. Upon testing, it was determined the Sciteq does not really turn off the output, it only attenuates the output by 30 dB.

4.3.5   Uplink Synchronization

The most significant modification to the HSC was the addition of support for uplink synchronization. Towards that end, three commands were added. The first was the ULSYNC command which causes the HSC to enter uplink synchronization mode. In this mode, the second command, FCALC, is used to calculate one frame's worth of the hopping frequencies for the uplink synchronization probes. Finally, the ULGO command is used to switch rapidly between the various probe frequencies.

Since there are now two types of synchronization (uplink and downlink) the old SYNC and GO commands were renamed to DLSYNC and DLGO.

5.   Conclusions

5.1   Summary

The author's task was to design and build a HSC to drive a Comstron or Sciteq synthesizer for a frequency hopped communications system. This was accomplished and the HSC was successfully tested during the Skynet downlink trials.

Flexibility was a key aspect of the HSC design. One HSC can be used in any of the four locations within a hopping system that use a synthesizer. The use of a digital signal processor and programmable read-only memories permit simple modification of software algorithms such as random number generation, frequency calculations, or synthesizer frequency word formatting. The use of an EPLD allows simple and rapid modification of the hardware interface.

The EPLDs were found to be very useful in replacing the numerous small integrated circuits normally required for any design. The design and programming of the EPLD were done on a personal computer simplifying design changes. Substituting one EPLD for all the usual support integrated circuits meant rapid development of the hardware as well as easy customization of the hardware interface which simplified the software requirement.

The HSC was successfully tested and, as predicted, the processor was idle for about one-half of the time. At maximum speed, the HSC will support hop rates as high as 25000 hops/s.

5.2   Future Work

It is important that Section 3.5, which deals with timing considerations, be read prior to any attempt to substitute faster components or otherwise modify the hardware.

The HSC was designed to be flexible and capable of expansion. It is now fully functional with plenty of capacity remaining on the board. 85% of the ROM and RAM are unused while the EPLD has 20% of its capability available. There is also board space for additional interface logic and another set of ROM or RAM chips. The processor itself is working at slightly over half of its capability.

The HSC could accommodate, with only firmware changes, different synthesizers. The HSC currently supports the Comstron and Sciteq synthesizers.

Currently, the random number generation is done by a multiplicative linear congruential pseudo-random generator that has been shortened to repeat after approximately four seconds. While this is good for development and testing of link hardware, there are two simplifications that would have to be changed to do a full test. The first change is to allow the random sequence to achieve its full length. This would be accomplished by removal of the code that forces the sequence to reinitialize after four

seconds. The other change to consider is the type of random number generator. There is no known way of jumping ahead in time for the multiplicative linear congruential generator. In the current software, each intermediate random number is generated - a time consuming process for long sequences. Another type of random number generator would be required such as a pseudo-random sequence generator. This would require the rewriting of the three RAN routines (RAN - generates random number, RANINI - initialization, and RANJMP - move forward in the random sequence). The serial port interface is currently unused, but is available to interface to a hardware pseudo-random sequence generator.

References

[1]   D.G. Carta, "Two Fast Implementations of the 'Minimal Standard' Random Number Generator", *Communications of the ACM*, Vol. 33, No. 1, pp. 87-88, January 1990.

[2]   R.D. Addison and W.R. Seed, *Implementation of an EHF Frequency-hopping Simulator*, DREO Report 1279, December 1995.

[3]   "Application Brief 89: Minimizing Output Switching Noise in Altera EPLDs", *Applications Handbook*, Altera Corporation, 1992.

[4]   *Operation and Maintenance Manual; Series FS2000 Frequency Synthesizer*, Comstron Corp., 1987.

[5]   *Operation and Maintenance Manual; Series FS2000A/B Frequency Synthesizer*, Comstron Corp., 1987.

[6]   *Operating Instructions: VDS-2G-469 Frequency Synthesizer*, Sciteq Electronics Inc., 1989.

[7]   *Second-Generation TMS320 User's Guide*, Texas Instruments Inc., 1987.

[8]   *User Manual for DT2817*, Data Translation Inc., 1988.

[9]   R. Addison, *Hopping Synthesizer Controller*, DREO Technical Note 90-17, August 1990.

[10]   P. da Sylva, *Hopping Synthesizer Controller Manual Update*, Applied Silicon Inc., 1991.

[11]   R. Addison, "Hopping Synthesizer Controller Documentation Erratum", Memorandum, Defence Research Establishment Ottawa, 1 March 1996.

[12]   J.D. Lambert, *DREO/CRC Joint Data Link Standard for Low Data Rate Service to EHF Ground Terminal and Payload Simulators*, DREO Report 1069, February 1991.

Appendix A: HSC User's Guide

1.   **General**

This appendix covers the information required to install, configure and use the Hopping Synthesizer Controller board. The latter portion consists of a description of the modes of operation and some examples of commanding the board through the Synchronization Controller Interface. The examples given are: initialization of the board, reading onboard RAM values, uplink and downlink synchronization, and setting a fixed frequency. Appendix B contains the command reference. Details of the interfaces can be found in Appendix C.

2.   **Installation**

This board drives a Comstron or Sciteq frequency synthesizer. A cable of no longer than three feet should connect the synthesizer to the 50-pin P1 connector on the back of the HSC. Interface and cable details are described in Appendix C.

If the HSC is to be controlled (not in stand-alone operation) then a cable is required to connect the Synchronization Controller to the 26-pin P2 connector on the HSC. In stand-alone operation, P2 is left unconnected. The current version of the software does not enable stand-alone operation for any of the possible configurations.

For FSK/Channel selection, a cable is required from the data source (Frame Formatter) to the 10-pin P3 connector on the back of the HSC. If no FSK/Channel selection is desired, then a stub is required on this connector. Details on this interface and the stub can be found in Appendix C.

The Serial Port Interface (P4) is not used in the current version of the software and should be left unconnected.

Once the cables have been connected, the HSC should be configured using the DIP switches.

3.   **Configuration**

At the bottom of the HSC a quad-wide DIP switch (SW1) is accessible. This switch is used to configure the board for operation in any position in the simulator. The details are given in Table A1.

Table A1. Configuration DIP switch settings.

Switch	Selects	Position	TTL Value	Value
SW1-1	Synthesizer Type	Closed/On Open/Off	0 1	Sciteq Comstron
SW1-2	Hop Clock Source	Closed/On Open/Off	0 1	External (on connector P2) Internal
SW1-3	Link Type	Closed/On Open/Off	0 1	Downlink Uplink
SW1-4	HSC Location	Closed/On Open/Off	0 1	Payload Ground Terminal

The four switches are described in the following paragraphs.

Synthesizer Type: The HSC can control two types of synthesizer. The two options are the Comstron synthesizer and the Sciteq synthesizer. The synthesizer type controls the computation and formatting for the frequency output and the use of a doubler on the output of the Sciteq.

Hop Clock Source: The normal source of the hop clock is externally from the Sync Controller interface (P2). For debugging or for stand-alone operation, the switch can be positioned to internal hop clock.

Link Type: This switch is used with the HSC Location switch to determine which end of which link is being serviced. There is a different set of default frequency parameters for the uplink and downlink.

HSC Location: This switch is used with the Link Type switch to determine which end of which link is being serviced. It can be in the ground terminal or payload positions. Different default parameters are used in each location.

Each of the four possibilities (uplink/downlink, payload/ground terminal) configurable by the last two switches has a different set of default values including hopping bandwidth, FSK modulation, base, offset and LO frequencies.

4.   **Hopping Synthesizer Controller Modes of Operation**

The processor has four distinct modes of operation. At power up, the processor is normally in STOP mode. The regular sequence would be to load in any required values, then command it into DLSYNC or ULSYNC modes and finally, when synchronization is achieved, command it into RUN mode. For a standalone hopper, the processor would power up in RUN mode. The details of the four modes follow:

STOP Mode
In this mode, the processor is only waiting for commands. There are no updates to the

26

synthesizer and there are no frequency calculations. This is the normal powerup mode. If STOP is commanded while in DLSYNC, ULSYNC or RUN modes, the HSC will stop hopping and leave the synthesizer at the last frequency.

In STOP mode, only LOAD commands, DEFAULTS command, mode commands and debug commands can be given. DLGO, ULGO and FCALC commands will be ignored. The command STOP when already in STOP mode is ignored. This feature is exploited during initialization by sending multiple STOP commands. (For more details see the Section 7.1 of this appendix) The response time to commands in this mode is less than 10 μs.

DLSYNC Mode

This mode is used when acquiring downlink coarse synchronization. The HSC waits in a tight loop for commands to switch to precomputed frequencies. In this mode, only the DLGO, RUN and STOP commands are allowed. To achieve fast response time, the command decoding is simplified so any other commands will give indeterminate results. The response time to commands in this mode is less than 3 μs.

RUN Mode

The processor should be put into RUN mode once synchronization has been achieved. A new hop frequency is computed and sent to the synthesizer each hop. In this mode, only the STOP command can be used. All other commands (including the RUN command) will be ignored. The response time to commands in this mode is less than one hop. There will only be one command processed per hop.

ULSYNC Mode

This mode is used when acquiring uplink coarse synchronization. The HSC waits in a tight loop for commands to precompute the frequencies and to rapidly switch among these frequencies. In this mode, only the ULGO, FCALC, RUN and STOP commands are allowed. To achieve fast response time, the command decoding is simplified so any other commands will give indeterminate results. The response time to commands in this mode is less than 3 μs.

5.   **Time Restrictions for Start Times**

For both uplink and downlink synchronization, the requirement for fast switching of frequencies necessitates precomputing the hops. Both the propagation of the random number generator and the precomputing of hops take significant time. This section examines the minimum calculation times to aid in selection of a start time that ensures all calculations are complete.

5.1   Start Time for Downlink Coarse Synchronization

The calculations for the start time for downlink coarse synchronization will be done to a resolution of one hop. For the random number propagation, at least 3 μs is needed for each hop from time 0.

$$T_{DLstart} \geq T_{current} + \frac{3\,\mu s}{t_{hop}} T_{DLstart}$$

where: (15)

$T_{DLstart}$ is time of first used downlink sync hop [hops]

$T_{current}$ is current time [hops]

t_{hop} is the duration of one hop [μs]

Solving for $T_{DLstart}$:

$$T_{DLstart} \geq \frac{T_{current}}{\left(1 - \frac{3\,\mu s}{t_{hop}}\right)}$$ (16)

For Skynet trials, use $t_{hop} \geq 53$ μs, which gives the following guideline to ensure that the HSC has enough time to compute in advance:

$$T_{DLstart} \geq 1.06\, T_{current}$$ (17)

The earliest start time should be after the current time and include an additional 6% for computation time. In a system where not all sync hops are assigned to the ground terminal, the start time should be the first assigned hop after the earliest start time. See Section 7.2 for an example.

5.2   Start Time for Uplink Coarse Synchronization

The calculations for the start time for uplink coarse synchronization will be done to a resolution of one hop, but will use key times on frame boundaries. Therefore instead of using the current time, the time of the next frame (after the current) will be used to start the calculations. For the calculations, at least 3 μs per hop is needed for the random number generator (which starts at time 0). An additional frame of time should be allowed before the start frame to calculate the probes for the start frame.

$$T_{ULstart} \geq T_{next} + \frac{3\,\mu s}{t_{hop}} T_{ULstart} + N_{hop/frame}$$

where: (18)

$T_{ULstart}$ is time of hop 0 of the start frame [hops]

T_{next} is time of hop 0 of the frame after the current frame [hops]

t_{hop} is the duration of one hop [μs]

$N_{hop/frame}$ is the number of hops per frame

Solving for $T_{ULstart}$:

$$T_{ULstart} \geq \frac{T_{next} + N_{hop/frame}}{\left(1 - \frac{3\ \mu s}{t_{hop}}\right)}$$ (19)

For Skynet trials, use $t_{hop} \geq 53\ \mu s$, which gives the following guideline to ensure that the HSC has enough time to compute in advance (note that all times must be on frame boundaries and are in units of hops):

$$T_{ULstart} \geq 1.06\left(T_{next} + N_{hop/frame}\right)$$

or

$$T_{ULstart} \geq 1.06\, T_{2^{nd}\ next}$$ (20)

where: $T_{2^{nd}\ next}$ is time of hop 0 of the second frame
after the current frame [hops]

The earliest start frame should be at least one full frame beyond the time of the next frame start and include an additional 6% for computation time. In a system where not all probes are assigned to the ground terminal, the start frame should be the first frame with assigned probes after the earliest start frame. See Section 7.3 for an example.

5.3   Start Time for Run

If the HSC is used in the payload, there is no requirement for synchronization to occur prior to a switch to RUN mode. To ensure the random number propagation calculations are complete before the chosen start time, it is necessary to allow computation time. Note that a switch to RUN mode after synchronization does not require this delay (see Section 7 of this appendix). The calculations for the start time for RUN mode will be done to the resolution of one hop. For the random number propagation, at least 3 μs is needed for each hop from time 0.

$$T_{RUNstart} \geq T_{current} + \frac{3\ \mu s}{t_{hop}} T_{RUNstart}$$

where: (21)
$T_{RUNstart}$ is time to switch to RUN mode [hops]
$T_{current}$ is current time [hops]
t_{hop} is the duration of one hop [μs]

29

Solving for $T_{RUNstart}$:

$$T_{RUNstart} \geq \frac{T_{current}}{\left(1 - \frac{3\,\mu s}{t_{hop}}\right)} \qquad (22)$$

For Skynet trials, use $t_{hop} \geq 53$ μs, which gives the following guideline to ensure that the HSC has enough time to compute in advance:

$$T_{RUNstart} \geq 1.06\,T_{current} \qquad (23)$$

The earliest start time should be after the current time and include an additional 6% for computation time. The start time can be any hop after and including the earliest start time.

6.   **Startup Sequence**

For diagnostic reasons, the HSC tests the interface lines after power-up. It does this by putting out the version number of the program, cycling each of the driven digits through the sequence 1 to 9 and by finally driving the synthesizer to mid-band of the hopping bandwidth. On the Comstron synthesizer, the version number and digits can be seen on the display. On some versions of the Comstron, due to the limited display on the upper digit, the number only cycles from 0 to 3 on the upper digit. Table A2. shows the startup sequence that is visible on the display of the Comstron synthesizers. The Sciteq synthesizer does not have a display and hence the startup sequence cannot be used to the same end. However, both the Sciteq and the Comstron are at mid-band after the power-up sequence is complete.

Table A2. Startup sequence on Comstron synthesizers.

Startup Sequence	Comstron FS2000B	Comstron FS2000	Remarks
Version Number	9605010300.0	1605010300.0	V960501.03, Version 3 of 01 May 96
Display/Cable Check	1111111100.0 2222222200.0 3333333300.0 4444444400.0 5555555500.0 6666666600.0 7777777700.0 8888888800.0 9999999900.0	1111111100.0 2222222200.0 3333333300.0 0444444400.0 1555555500.0 2666666600.0 3777777700.0 0888888800.0 2999999900.0	4, 8 GHz bits not used on the FS2000
Midband	0670000000.0	0670000000.0	For uplink Comstron configuration only (otherwise a different frequency) Equivalent to LOAD HOP 800000h

30

7.   Examples

In the this section, examples of HSC initialization, downlink synchronization, uplink synchronization and setting a fixed frequency are provided. Each example has a background portion which specifies the values used in the example, and the steps necessary to achieve the operation. The steps detail each command including the hexadecimal values to which they correspond.

7.1   Initialization

Initialization Background

This is an example of a typical initialization operation. It is assumed that the HSC is in an unknown mode, with unknown defaults, and possibly in the middle of a command. The following assumptions are made:

RF - The RF input range is 41 GHz to 42.2 GHz with a 0.1 GHz guard band on each end. The downconversion before the synthesizer is 40 GHz. The Comstron synthesizer will be used (no doubling). Doppler will not be used. FSK modulation will not be used.

Time - Arbitrarily assume that the start time is hop 45 (range 0 to 319) of frame 17 (frame numbers start at 0)

Initializing consists of getting the processor to a known state (STOP mode), loading the values and the loading the time. The necessary commands are listed below.

Initialization Step 1 - Force HSC to STOP Mode

To ensure that the processor was not waiting for data words, one must send the STOP command three times if the state of the processor is unknown. It is possible that a LOAD command, without associated data, was sent prior to this point. The two extra commands will provide the data for the LOAD command (completing the execution of the pending LOAD command) and the third STOP will be executed as a normal STOP command.

Operation	Hex Words (in order)	Remarks
STOP	0000	
STOP	0000	
STOP	0000	

31

Initialization Step 2 - Load Parameters

Operation	Hex Words (in order)	Remarks
LOAD BWSCALE 1 GHz	0306,4B40,004C	5000000=4C4B40h [100 Hz] (One-half of hopping bandwidth less guard bands is 0.5 GHz)
LOAD FCSPACE 0	030A,0000,0000	0 (No FSK modulation)
LOAD BASE 41 GHz	030C,1A80,1870	410000000=18701A80h [100 Hz]
LOAD OFFSET 0.1 GHz	030E,4240,000F	1000000=F4240h [100 Hz]
LOAD DOPFACT 0	0310,0000,0000	0 (No Doppler)
LOAD LOCOM 40 GHz	0312,8400,17D7	400000000=17D78400h [100 Hz]
LOAD LOSCI 40 GHz	0318,8400,17D7	400000000=17D78400h [100 Hz] (Not used in this configuration)
LOAD FLAGS 0	031A,0000,0000	0 (No Sciteq doubler)

Initialization Step 3 - Load Current Time

Hop 45 of frame 17 corresponds to hop 5485 (17 x 320 + 45).

Operation	Hex Words (in order)	Remarks
LOAD TIMELO 5485	0314,156D,0000	5485=156Dh [hops] (also automatically sets TIMEHI to 0)

7.2   Downlink Synchronization

Downlink Synchronization Background

This is an example of a typical downlink synchronization operation. It is assumed that the downlink HSC was initialized with default or unknown parameters and is in STOP mode. The following assumptions are made:

Sync Hops - These are BPSK modulated with a burst cycle of 16 frames and for this example, the terminal is using the first frame of every cycle to transmit sync hops. Within a sync hop burst, the first four even hops are used. So the downlink sync hop assignment is once every sixteen frames on multiples of 16 (ie: 0,16,32,48,...). The first burst is used for detection and estimation, with the second burst used for verification. A burst starts on the 288th hop of a 320-hop frame.

RF - The RF input range is 41 GHz to 42.2 GHz with a 0.1 GHz guard band on each end. The downconversion before the synthesizer is 40 GHz. The Comstron synthesizer will be

used (no doubling). Doppler will not be used.

Time - Arbitrarily assume that the estimated downlink time is now hop 12 (range 0 to 319) of frame 43 (frame numbers start at 0)

Fig. A1. shows graphically the timing of the various commands used in the downlink synchronization example.
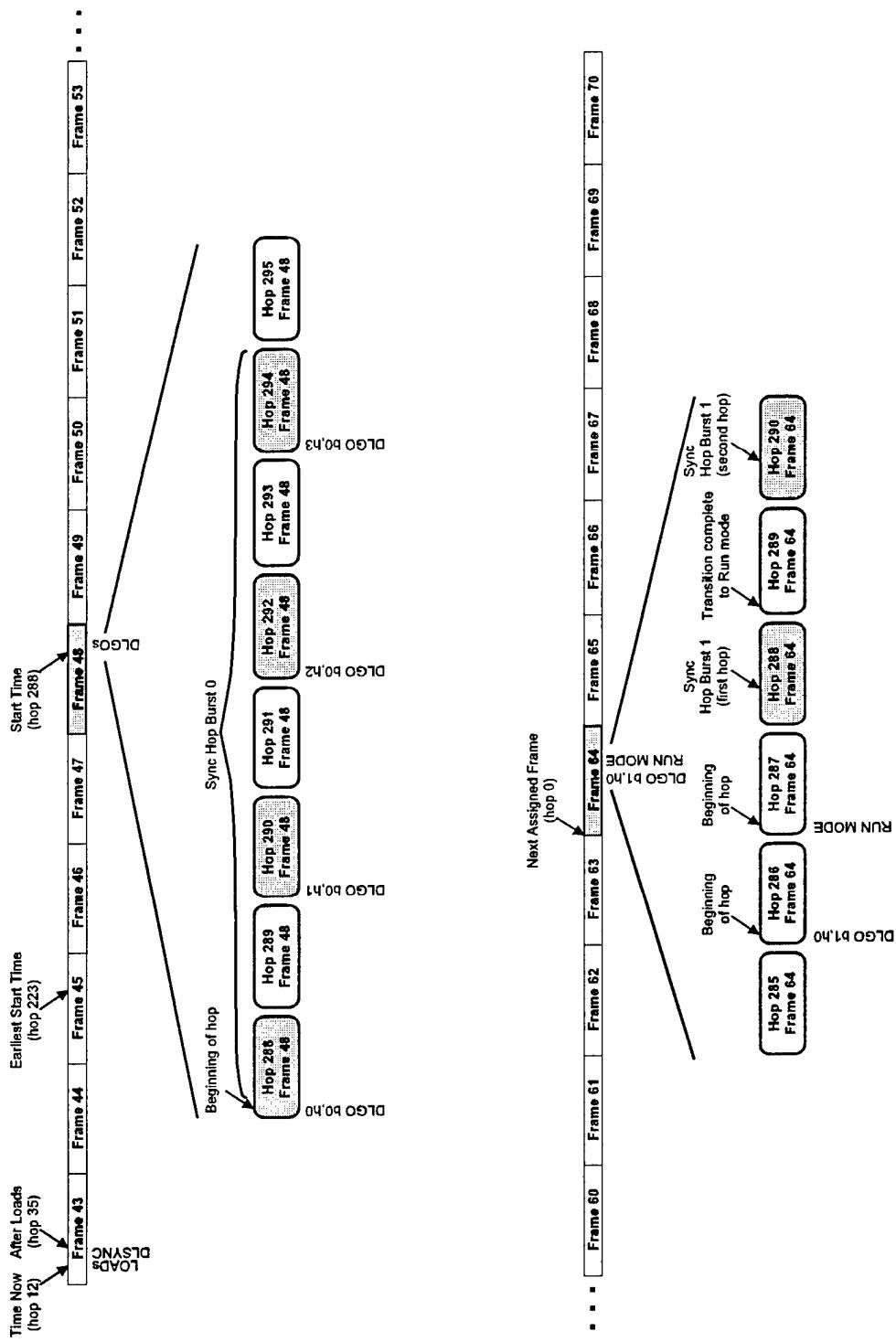
Fig. A1. Command timing for downlink synchronization example.

34

Downlink Synchronization Step 1 - Load Downlink Parameters

Operation	Hex Words (in order)	Remarks
LOAD BWSCALE 1 GHz	0306,4B40,004C	5000000=4C4B40h [100 Hz] (One-half of hopping bandwidth less guard bands is 0.5 GHz)
LOAD FCSPACE 0	030A,0000,0000	0 (No FSK modulation)
LOAD BASE 41 GHz	030C,1A80,1870	410000000=18701A80h [100 Hz]
LOAD OFFSET 0.1 GHz	030E,4240,000F	1000000=F4240h [100 Hz]
LOAD DOPFACT 0	0310,0000,0000	0 (No Doppler)
LOAD LOCOM 40 GHz	0312,8400,17D7	400000000=17D78400h [100 Hz]
LOAD LOSCI 40 GHz	0318,8400,17D7	400000000=17D78400h [100 Hz] (Not used in this configuration)
LOAD FLAGS 0	031A,0000,0000	0 (No Sciteq doubler)

Downlink Synchronization Step 2 - Load Downlink Start Time

These calculations are based on Section 5.1 of this appendix. The estimated time before any of the loads takes place is hop 12 of frame 43. The estimated time now after the loads is hop 35 of frame 43. In hops, the estimated time now is 13795 (43 x 320 + 35). 6% is added (827.7 hops goes to 828 hops) to get 14623 hops. The earliest start time is hop 223 of frame 45. The next coarse probe assigned after that is hop 288 of frame 48 (multiple of 16). Therefore the start time is hop 15648 (48 x 320 + 288).

Operation	Hex Words (in order)	Remarks
LOAD TIMELO 15648	0314,3D20,0000	15648=3D20h [hops] (also automatically sets TIMEHI to 0)

Downlink Synchronization Step 3 - Switch to Downlink Sync Mode

Operation	Hex Words (in order)	Remarks
DLSYNC	4000	

The operation is complete when the Sync status line is set. Once the calculations are complete, the synthesizer will be set to the first downlink sync hop frequency.

35

Downlink Synchronization Step 4 - Setup for First Hop

One downlink sync hop burst consists of four hops with each hop spaced at intervals of two hops. Wait for the estimated start of the first hop then switch to the probe frequency.

Operation	Hex Words (in order)	Remarks
DLGO Burst 0, Hop 0	0060	Any time up to, but no later than, hop 288 frame 48 LSB (burst x 4 + hop) x 2

Downlink Synchronization Step 5 - Setup for Remaining Hops of First Burst

Switch to the remaining hops of the first burst in turn after skipping one hop each time.

Operation	Hex Words (in order)	Remarks
DLGO Burst 0, Hop 1	0062	At hop 290 frame 48 LSB (burst x 4 + hop) x 2
DLGO Burst 0, Hop 2	0064	At hop 292 frame 48
DLGO Burst 0, Hop 3	0066	At hop 294 frame 48

Downlink Synchronization Step 6 - Setup for Verification and Tracking

Assume that the detection on the first burst is successful and the downlink time has been successfully estimated. The burst starting at hop 288 of frame 64 will be used for verification. The HSC should be switched into RUN mode for the verification and subsequent fine synchronization acquisition and tracking. The mode switch should occur on the hop preceding the first hop of the second burst. Since it takes one hop for RUN mode transition to be completed, the DLGO command will be used to setup the first hop before RUN mode is commanded - this can be done anytime between the two bursts. The sync line will be reset when the RUN command is received.

Operation	Hex Words (in order)	Remarks
DLGO Burst 1, Hop 0	0068	Any time up to, but no later than, hop 286 frame 64 LSB (burst x 4 + hop) x 2
RUN	8000	At hop 287 frame 64

36

Downlink Synchronization Failure

If downlink synchronization is lost, the detection phase fails or can not be verified, then a change to STOP mode should be made. Once the sync status bit is reset, restarting at Downlink Synchronization Step 2 to will allow another attempt with a new start time.

Operation	Hex Words (in order)	Remarks
STOP	0000	

7.3 Uplink Synchronization

Uplink Synchronization Background

This is an example of a typical uplink coarse synchronization operation. It is assumed that the uplink HSC was initialized with default or unknown parameters and is in STOP mode. The following assumptions are made:

Sync Probes - Uplink coarse sync probes are in transmitted in channel 5 (range 0-31), FSK bin 3 (range 0-7) with an FSK bin spacing of 36 kHz. The probing cycle is 4 frames and this terminal is assigned the first frame of every cycle for probing. So the uplink coarse sync probing assignment is once every four frames on multiples of 4 (ie: 0,4,8,12,...). The probe burst starts on the 288th hop of a 320-hop frame.

RF - The RF input range is 41 GHz to 42.2 GHz with a 0.1 GHz guard band on each end. The upconversion after the synthesizer is 40 GHz. The Comstron synthesizer will be used (no doubling). Doppler will not be used.

Time - Arbitrarily assume that the estimated downlink time is now hop 12 (range 0 to 319) of frame 23 (frame numbers start at 0).

Fig. A2. shows graphically the timing of the various commands used in the uplink synchronization example.
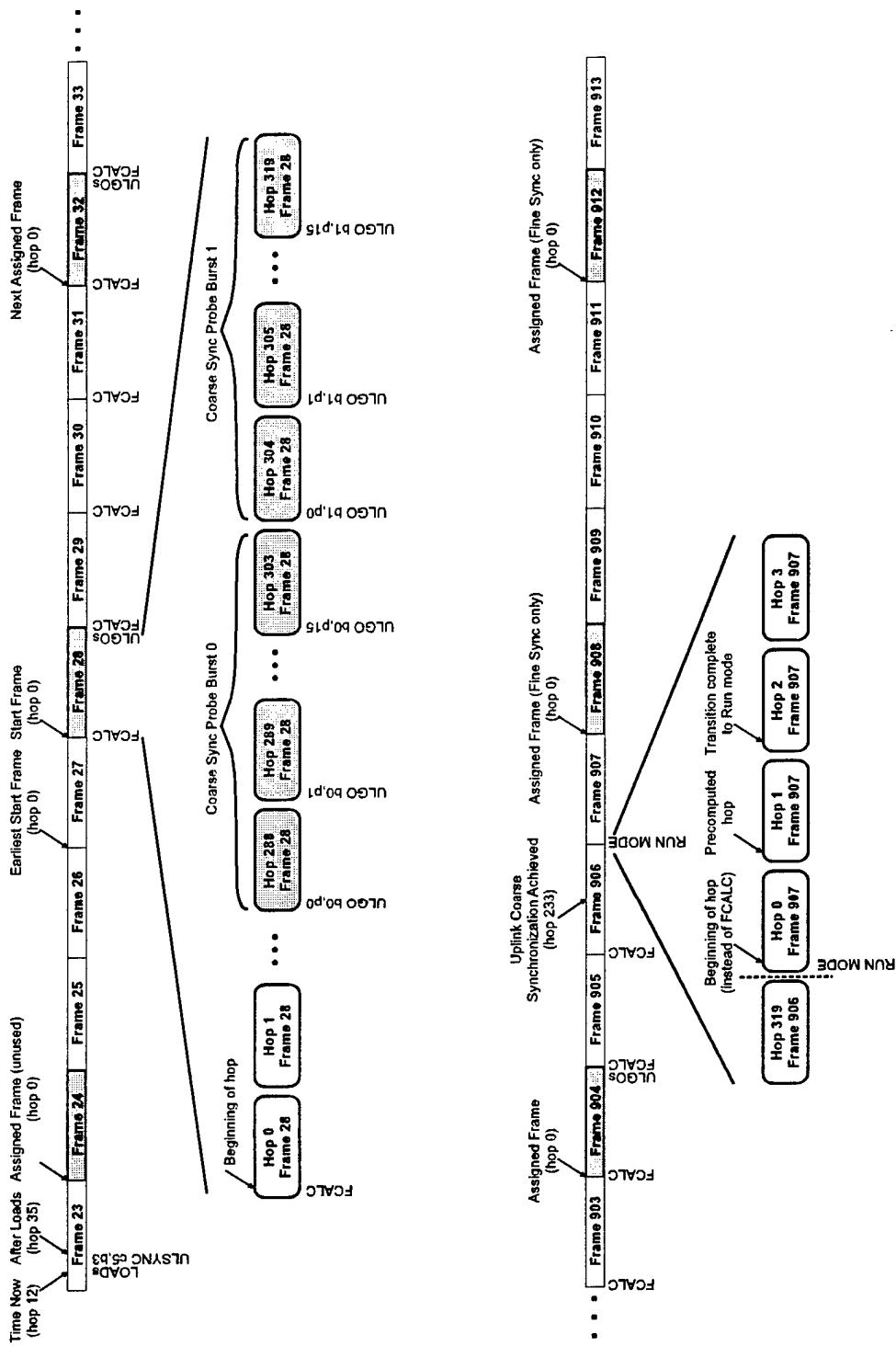
Fig. A2. Command timing for uplink synchronization example.

38

Uplink Synchronization Step 1 - Downlink Sync

Before uplink synchronization can be achieved, one must first achieve downlink synchronization. The previous example demonstrates this process.

Uplink Synchronization Step 2 - Load Uplink Parameters

Operation	Hex Words (in order)	Remarks
LOAD BWSCALE 1 GHz	0306,4B40,004C	5000000=4C4B40h [100 Hz] (One-half of hopping bandwidth less guard bands is 0.5 GHz)
LOAD FCSPACE 36 kHz	030A,0168,0000	360=168h [100 Hz]
LOAD BASE 41 GHz	030C,1A80,1870	410000000=18701A80h [100 Hz]
LOAD OFFSET 0.1 GHz	030E,4240,000F	1000000=F4240h [100 Hz]
LOAD DOPFACT 0	0310,0000,0000	0 (No Doppler)
LOAD LOCOM 40 GHz	0312,8400,17D7	400000000=17D78400h [100 Hz]
LOAD LOSCI 40 GHz	0318,8400,17D7	400000000=17D78400h [100 Hz] (Not used in this configuration)
LOAD FLAGS 0	031A,0000,0000	0 (No Sciteq doubler)

Uplink Synchronization Step 3 - Load Uplink Start Time

These calculations are based on Section 5.2 of this appendix. The estimated time before the loads is hop 12 of frame 23. After the loads, the estimated time is hop 35 of frame 23. The next frame is frame 24 and the second next frame start is hop 0 of frame 25. 6% is added (at least 1.5 frames goes to 2 frames) for calculation time to get earliest start at hop 0 of frame 27. The next coarse probe assigned frame is hop 0 of frame 28 (multiple of 4), so the start time is hop 8960 (28 x 320).

Operation	Hex Words (in order)	Remarks
LOAD TIMELO 8960	0314,2300,0000	8960=2300h [hops] (also automatically sets TIMEHI to 0)

39

Uplink Synchronization Step 4 - Switch to Uplink Sync Mode

Operation	Hex Words (in order)	Remarks
ULSYNC Channel 5, Bin 3	502B	43=2Bh FSK/channel parameter LSB (channel x 8 + bin)

The operation is complete when the Sync status line is set.

Uplink Synchronization Step 5 - Calculate Probes for Start Frame

Issue the FCALC command on hop number 0 of the assigned start frame (frame 28). The calculations will be complete before the probing starts.

Operation	Hex Words (in order)	Remarks
FCALC	5800	Calculate for frame 28

Uplink Synchronization Step 6 - Transmit First Probe

The probes in one coarse uplink assignment are sent in two bursts of 16 probes. Each probe burst has a different time hypothesis. Wait for the start of the first probe (hop 288 of frame 28) for the first timing hypothesis and then switch to the probe frequency.

Operation	Hex Words (in order)	Remarks
ULGO Burst 0, Probe 0	0200	LSB (burst x 16 + probe) x 2

Uplink Synchronization Step 7 - Transmit Remaining Probes of First Burst

Switch to the remaining probes of first burst in turn at the beginning of the next 15 hops.

Operation	Hex Words (in order)	Remarks
ULGO Burst 0, Probe 1	0202	LSB (burst x 16 + probe) x 2
ULGO Burst 0, Probe 2	0204	
. . .	. . .	
ULGO Burst 0, Probe 15	021E	

40

Uplink Synchronization Step 8 - Transmit First Probe of Second Burst

At the start of the first probe (hop 304 of frame 28) for the second timing hypothesis switch to that probe frequency.

Operation	Hex Words (in order)	Remarks
ULGO Burst 1, Probe 0	0220	LSB (burst x 16 + probe) x 2

Uplink Synchronization Step 9 - Transmit Remaining Probes of Second Burst

Switch to the remaining probes of second burst in turn at the beginning of the next 15 hops.

Operation	Hex Words (in order)	Remarks
ULGO Burst 1, Probe 1	0222	LSB (burst x 16 + probe) x 2
ULGO Burst 1, Probe 2	0224	
. . .	. . .	
ULGO Burst 1, Probe 15	023E	

Uplink Synchronization Step 10 - Skip Unassigned Frames

No probes are transmitted during the next three frames because they are not part of this terminal's uplink coarse probe assignment in this example. However, in order to maintain the random number generator sequence, FCALC commands must still be issued for each frame. Each FCALC command should be invoked near the beginning of the respective frame (but no more than one frame calculation should be commanded per frame).

Operation	Hex Words (in order)	Remarks
FCALC	5800	Skip frame 29
FCALC	5800	Skip frame 30
FCALC	5800	Skip frame 31

Uplink Synchronization Not Yet Finished

If uplink coarse sync has not be achieved, return to Uplink Synchronization Step 5 to continue on the next probe cycle.

41

Uplink Synchronization Step 11 - Setup for Fine Synchronization and Tracking

Assume that uplink coarse synchronization is achieved on hop 233 of frame 906. The HSC should be switched into RUN mode for the fine synchronization procedure. The mode switch should occur instead of the next FCALC which is at the beginning of frame 907. RUN mode must be commanded after the start of the first hop (hop number 0) of a frame but before the second hop (hop number 1) of frame 907. The HSC will load the frequency for the second hop and will have completed the transition to RUN mode by the third hop (hop number 2) of frame 907. The Sync status bit will be reset in RUN mode.

Operation	Hex Words (in order)	Remarks
RUN	8000	between hop 0 and hop 1 of frame

Uplink Synchronization Failure

If uplink synchronization is lost, then switch to STOP mode. When the STOP mode change is complete, the Sync status bit will be reset. Restarting is achieved by then proceeding from Uplink Synchronization Step 3 with a new start time.

Operation	Hex Words (in order)	Remarks
STOP	0000	

7.4   Setting a Fixed Frequency

Fixed Frequency Background

While debugging the RF chain it is often useful to be able to stop hopping and have the synthesizer go to a specific frequency. This example demonstrates two ways to do this. The first way commands the synthesizer directly (called Direct Fixed Frequency), whereas the second way selects a frequency from within the hopping bandwidth (called Hop Number Fixed Frequency). It is assumed that the downlink HSC is in STOP mode and that the Comstron synthesizer is being used. The frequency desired is 1.4 GHz.

Direct Fixed Frequency Step 1 - Immediate Synthesizer Latching

To set the latches directly with the binary value sent to the synthesizer, it is first necessary to ensure that the synthesizer changes immediately to the new frequency regardless of the hop clock.

Operation	Hex Words (in order)	Remarks
CHANGE IMMEDIATE	0501	

Direct Fixed Frequency Step 2 - Set the Frequency

Next the latches that feed the synthesizer must be set with the desired binary value. Since the Comstron synthesizer uses BCD and the units are 100 Hz, 14000000h will generate 1.4 GHz. (For the Sciteq synthesizer, the calculations are detailed in Section 2.4.3. in the main report.)

Operation	Hex Words (in order)	Remarks
LOAD LATCH 1.4 GHz	0300,0000,1400	335544320 = 14000000h

This step can be repeated with different values for the latch to select different frequencies.

Direct Fixed Frequency Step 3 - Restore Normal Synthesizer Latching

Finally, the normal synthesizer latching should be restored.

Operation	Hex Words (in order)	Remarks
CHANGE HOP	0500	

Hop Number Fixed Frequency Step 1 - Immediate Synthesizer Latching

To program the synthesizer using the hop number, it is first necessary to ensure that the synthesizer changes immediately to the new frequency regardless of the hop clock.

Operation	Hex Words (in order)	Remarks
CHANGE IMMEDIATE	0501	

Hop Number Fixed Frequency Step 2 - Load Frequency Parameters

It is desired to set a frequency of 1.4 GHz. Choose, arbitrarily, a hopping bandwidth of 1 to 2 GHz with no guard bands to allow setting other frequencies near the desired one later.

43

Operation	Hex Words (in order)	Remarks
LOAD BWSCALE 1 GHz	0306,4B40,004C	5000000=4C4B40h [100 Hz] (One-half of hopping bandwidth less guard bands is 0.5 GHz)
LOAD BASE 1 GHz	030C,9680,0098	10000000=989680h [100 Hz]
LOAD OFFSET 0	030E,0000,0000	0 [100 Hz]
LOAD DOPFACT 0	0310,0000,0000	0 (No Doppler)
LOAD LOCOM 0	0312,0000,0000	0 [100 Hz]
LOAD LOSCI 0	0318,0000,0000	0 [100 Hz] (Not used in this configuration)
LOAD FLAGS 0	031A,0000,0000	0 (No Sciteq doubler)

Hop Number Fixed Frequency Step 3 - Select Frequency

Now the hop number appropriate for the frequency must be calculated. The lower edge of the band is 1 GHz and corresponds to a LOAD HOP 0. The upper edge is 2 GHz and corresponds to a LOAD HOP 16777215 (FFFFFFh). Interpolating, 1.4 GHz corresponds to LOAD HOP 6710886 (666666h).

Operation	Hex Words (in order)	Remarks
LOAD HOP 6710886	0304,6666,0066	6710886 = 666666h = 40% of hopping bandwidth

This step can then be repeated with any value of hop number to select any frequency in the hopping bandwidth.

Hop Number Fixed Frequency Step 4 - Restore Normal Synthesizer Latching

Finally, the normal synthesizer latching should be restored.

Operation	Hex Words (in order)	Remarks
CHANGE HOP	0500	

44

7.5   Reading HSC Memory

Read Background

During debugging, it is often useful to verify the parameters of the HSC or to check the internal hop count. To examine internal RAM values within the HSC, the READ command can be used. It is assumed that the HSC has been initialized with the default parameters, possibly modified using LOAD commands and is currently in STOP mode. For this example, the BWSCALE value at address 0306H is to be read. Note that the Sync line is used to return the data. This line also controls the synthesizer latching. Because of this, the synthesizer may change frequencies during the READ command.

Read Step 1 - Read Command

The READ command causes the HSC to load the appropriate RAM value into the accumulator in preparation to be shifted out one bit at a time on the Sync line.

Operation	Hex Words (in order)	Remarks
READ 306h	2306	Read BWSCALE at address 306h

Read Step 2 - Get Bit 15 (MSB)

During a read operation, when any value (other than 0 for the STOP command) is sent, the HSC sets the NReady line, then puts out the next bit on the Sync line and clears the NReady line. The first bit out is the most significant bit - bit 15 of the value. The simplest value to send is a repeat of READ command.

Operation	Hex Words (in order)	Remarks
READ 306h	2306	

When ready (NReady is clear), the Sync line has the 15th bit.

Read Step 3 - Get Remaining Bits (14-0)

Each time a command other than STOP is sent, the HSC outputs another bit on the Sync line (it is simplest to repeat the READ command). This should be done 15 more times to get the remaining bits.

45

Operation	Hex Words (in order)	Remarks
READ 306h	2306	Bit 14
READ 306h	2306	Bit 13
...	...	...
READ 306h	2306	Bit 0

Read Step 4 - End of Read

A STOP command is then used to end the READ and to return the HSC to STOP mode.

Operation	Hex Words (in order)	Remarks
STOP	0000	

After the STOP command is received the HSC forces the Sync line to 0. This has the same effect as a CHANGE HOP command.

Appendix B: HSC Command Reference

1. General

This appendix provides the detailed information on each command including the hexadecimal (hex) value, parameters required, HSC modes that support each command, description of the command, example of use and related commands. Also included in this appendix are command lists by category and by hexadecimal value for cross reference purposes.

2. Commands Listed By Category

2.1 Load Commands

These commands load the values that affect frequency computations and can only be used in STOP mode. Each LOAD consists of the command word followed by the lower 16-bit word and then the upper 16-bit word of a 32-bit longword. LOAD commands are not required unless values other than the defaults are needed. Note that time (TIMEHI and TIMELO) must be loaded before switching to DLSYNC, ULSYNC or RUN modes. The default values for these parameters are given in Table B1.

LOAD BASE	Load base frequency
LOAD BWSCALE	Load hopping bandwidth scale factor
LOAD DOPFACT	Load Doppler factor
LOAD FCSPACE	Load FSK tone bin spacing
LOAD FLAGS	Load configuration control flags
LOAD LOCOM	Load the Comstron up/downconversion LO value
LOAD LOSCI	Load the Sciteq up/downconversion LO value
LOAD OFFSET	Load the guard band frequency offset
LOAD TIMELO	Load lower 32 bits of time and clear upper 32 bits
LOAD TIMEHI	Load upper 32 bits of time

2.2 Synchronization Commands

Synchronization commands consist of the commands for switching to uplink or downlink synchronization modes, precomputing sync hops or probes, and rapidly switching between these precomputed frequencies. All synchronization commands consist of a 16-bit word. Where parameters exist, they form part of the 16-bit word. Synchronization commands are necessary to perform synchronization at the ground terminal. The payload does not require the use of these commands. ULSYNC and DLSYNC commands should only be used in STOP mode. DLGO, FCALC and ULGO should only be used in the appropriate synchronization mode. Note that FCALC is required to be used once per frame in ULSYNC mode prior to any ULGOs (even if none of the frequencies calculated is used).

DLSYNC	Switch to DLSYNC (downlink synchronization) mode
DLGO	Select downlink sync hop frequency
FCALC	Calculate a frame's worth of uplink sync probe frequencies
ULGO	Select uplink sync probe frequency
ULSYNC	Switch to ULSYNC (uplink synchronization) mode

Table B1. Default values for parameters.

Variable	GT Up	GT Down	Pay Up	Pay Down	Remarks
LATCH	-	-	-	-	(debug only)
SEED	-	-	-	-	(debug only)
HOPNUM	0	0	0	0	(debug only)
BWSCALE	250000	250000	250000	250000	50 MHz BW
FSKCHAN	0	0	0	0	(debug only)
FCSPACE	360	360	360	360	36 kHz spacing
BASE	445950000	445950000	76200000	76200000	44.6/7.6 GHz
OFFSET	0	0	0	0	No guard band
DOPFACT	0	0	0	0	No Doppler
LOCOM	439500000	439500000	40950000	40950000	44.0/4.1 GHz
TIMELO	0	0	0	0	Time = 0
TIMEHI	0	0	0	0	
LOSCI	431100000	431100000	40950000	40950000	43.1/4.1 GHz
FLAGS	0	0	0	0	No x2, RUN

2.3   Operating Commands

This is the last group of commands that is used in normal operations. Each command consists of a 16-bit word. The STOP command can be given in any mode. RUN can be used in STOP mode or at the completion of synchronization in either of the synchronization modes. DEFAULTS should only be used in STOP mode.

DEFAULTS	Select default values based on current switch configuration
RUN	Switch to RUN mode
STOP	Switch to STOP mode

2.4   Debugging Commands

Debugging commands are used in the development stage of the hardware or software systems. These commands include several LOAD commands that are of no use in normal operation. The LOAD commands use three 16-bit words - one for the LOAD itself and two for the data to be loaded. All other commands are contained in one 16-bit word. All debugging commands can only be used in STOP mode. Note that the READ command affects the Sync line and when terminated with a STOP command, has an implied CHANGE HOP command.

CHANGE HOP	Change synthesizer on rising edge of hop clock
CHANGE IMMEDIATE	Change synthesizer when latches are loaded
LOAD FSKCHAN	Load FSK Bin/Channel number
LOAD HOP	Load hop number, calculate and put frequency into latches
LOAD LATCH	Load HSC on-board latches directly
LOAD SEED	Load random number seed
READ	Read C25 internal RAM data

3.   **Commands by Hexadecimal Value**

Table B2. HSC Commands by Hexadecimal Value.

Hex Value	Command
0	STOP
60, 62, ... 66 68	DLGO Hop 0 Burst 0 - DLGO Hop 3 Burst 0 DLGO Hop 0 Burst 1
200, 202, ... 21E 220, 222, ... 23E	ULGO Probe 0 Burst 0 - ULGO Probe 15 Burst 0 ULGO Probe 0 Burst 0 - ULGO Probe 15 Burst 0
300 302 304 306 308 30A 30C 30E 310 312 314 316 318 31A	LOAD LATCH LOAD SEED LOAD HOP LOAD BWSCALE LOAD FSKCHAN LOAD FCSPACE LOAD BASE LOAD OFFSET LOAD DOPFACT LOAD LOCOM LOAD TIMELO LOAD TIMEHI LOAD LOSCI LOAD FLAGS
400	DEFAULTS
500 501	CHANGE HOP CHANGE IMMEDIATE
2000 - 2FFF	READ
4000	DLSYNC
5000 - 50FF	ULSYNC
5800	FCALC
8000	RUN

Note that for simplicity of command processing and fast response, the hexadecimal value for LOAD, DLGO and ULGO match the C25 address used to store the appropriate value. To allow a 32-bit word to be stored (in 16-bit memory), each of the hexadecimal values for these commands is separated by 2.

50

## 4.	Commands Listed Alphabetically

The following pages list the commands alphabetically with all the details of usage including an example for each command. With each command, the following fields are detailed:

Command:	This is the name of the command that is used in the documentation.

Hex Value:	The hexadecimal value that must be sent to the HSC to cause this command to be processed. Where a field with "X" occurs in this value, a parameter must be substituted. For example, the DLGO command has a value of 006X. The X must be replace by one of 0, 2, 4, 6 or 8 depending on which downlink hop is desired. This results in valid commands of 0060, 0062, 0064, 0066 or 0068.

Parameter:	This describes the parameter (if appropriate) for the command and how to calculate the value to be substituted (see Hex Value).

Valid Modes:	These are the modes of the HSC in which this command can be given. It is one of STOP, RUN, DLSYNC, ULSYNC or a special mode STOP (debugging). The latter indicates that the command must be issued in normal STOP mode, but is only useful for debugging purposes and should not be used during normal operation.

Description:	This includes a description of the command, method of use and valid ranges if appropriate.

Example:	The example is an illustrative use of the command. Where a command cannot be used by itself, the other necessary commands are included in the example. The example gives the command name with parameters, the hexadecimal values to be sent to the HSC to effect this command and clarifying remarks.

Related Commands:	This lists any commands that may be related, either commands that perform similar functions or commands that are used with the subject command.

Command: CHANGE HOP
Hex Value: 0500
Parameter: -
Valid Modes: STOP (debugging)

Description: This command returns the HSC to normal operation with respect to changing frequencies at the synthesizer. After this command is invoked, the synthesizer changes frequencies on the rising edge of the hop clock rather than upon latching of the frequency value.

Example:

Operation	Hex Words (in order)	Remarks
CHANGE HOP	0500	

Related Commands: CHANGE IMMEDIATE, LOAD HOP, LOAD LATCH

Command: CHANGE IMMEDIATE
Hex Value: 0501
Parameter: -
Valid Modes: STOP (debugging)

Description: This command is used during debugging to cause the synthesizer to change immediately when the onboard latches are written (such as after a LOAD HOP or LOAD LATCH command). Under normal operation, the synthesizer only changes on the rising edge of the hop clock.

Example:

Operation	Hex Words (in order)	Remarks
CHANGE IMMEDIATE	0501	

Related Commands: CHANGE HOP, LOAD HOP, LOAD LATCH

Command: DEFAULTS
Hex Value: 0400
Parameter: -
Valid Modes: STOP

Description: This command causes the default values to be reloaded from the read-only memory (ROM). This command should be used to ensure that all of the parameters have known initial values. This command is automatically invoked on powerup of the HSC.

Example:

Operation	Hex Words (in order)	Remarks
DEFAULTS	0400	

Related Commands: all LOADs

Command: DLGO <sync_hop>

Hex Value: 006X

Parameter:

The sync hop number portion of the command (X) is calculated using:

$$X = 2\left(4N_{Burst} + N_{Hop}\right)$$

where:

    X is the least significant 4 bits of the command (0-E)   (24)

    N_{Burst} is the burst number (0 for first 4 hops, 1 for the last)

    N_{Hop} is the sync hop number (0-3) within the burst

Valid Modes: DLSYNC

Description: This command is used to switch rapidly from one precomputed frequency hop to another. When the processor is commanded into DLSYNC mode, five sync hops are precomputed - four in the first burst and one in the second sync hop burst. Note that the DLSYNC command also causes the first sync hop (Burst 0, Hop 0) to be set, so the only use for DLGO Burst 0, Hop 0 is when restarting after a false alarm.

The Sync line is set upon completion of the calculations and should be checked prior to the use of DLGO commands.

Frequency changes will occur within 3 µs of the DLGO command.

Example:

Operation	Hex Words (in order)	Remarks
DLGO Burst 0, Hop 1	0062	

Related Commands: DLSYNC, ULGO

Command: DLSYNC
Hex Value: 4000
Parameter: -
Valid Modes: STOP

Description: This command causes the processor to precompute five sync hop frequency values based on the current time, sets the synthesizer to the first sync hop and then enters a tight loop waiting for DLGO commands or a change to RUN or STOP mode. DLGO commands within DLSYNC mode cause an immediate change in the synthesizer's frequency. The Sync line is set once all the computations are complete.

LOAD TIMELO and if required LOAD TIMEHI should be used to set the current time prior to the DLSYNC command. Section 5.1 of Appendix A details the method for choosing the desired time.

The first sync hop burst consists of four hops spaced one hop apart. The second burst starts 5128 hops after the start of the first burst.

Because of the speed needed to respond to DLGO commands, the processor's code has been streamlined to eliminate error checking. This means that illegal commands will cause the processor to enter an indeterminate state. Only RUN, STOP or DLGO commands should be used in DLSYNC mode.

Example:

Operation	Hex Words (in order)	Remarks
DLSYNC	4000	

Related Commands: DLGO, RUN, STOP, ULSYNC

56

Command: FCALC
Hex Value: 5800
Parameter: -
Valid Modes: ULSYNC

Description: Calculate one frames worth of probes in uplink sync mode. This command should be given during the first hop (hop number 0) of the frame.

This command must be used before any ULGO commands.

Because this command is the only way to advance the current time within ULSYNC mode, the command must be given for each frame even if the probe calculations are not used.

FCALC calculates from the third hop of the current frame to the second hop of the subsequent frame. The first two hops were computed either by the preceding FCALC or ULSYNC command.

Example:

Operation	Hex Words (in order)	Remarks
FCALC	5800	

Related Commands: ULGO, ULSYNC

57

Command: LOAD BASE <base_freq>

Hex Value: 030C,$XXXX_{least}$,$XXXX_{most}$

Parameter: Base frequency [units of 100 Hz]

Valid Modes: STOP

Description: This command loads a 32 bit value to be used as the base frequency. This is the lower edge of the allocated bandwidth. The lower guard band offset from this base frequency (to allow for Doppler and modulation) should be specified with the LOAD OFFSET command.

The base frequency can range in value from 0 to 999000000 (3B8B87C0h) corresponding to 0 Hz to 99.900 GHz.

Example:

Operation	Hex Words (in order)	Remarks
LOAD BASE 43.5 GHz	030C,92C0,19ED	435000000 = 19ED92C0h

Related Commands: LOAD BWSCALE, LOAD OFFSET

Command: LOAD BWSCALE <scale_factor>

Hex Value: 0306,XXXX$_{least}$,XXXX$_{most}$

Parameter: Hopping bandwidth scale factor [units of 200 Hz]
    (**Note this load is an exception, all others are in units of 100 Hz)**

Valid Modes: STOP

Description: This command loads the scale factor that determines the hopping bandwidth. This is the number that is multiplied by the random hop number to get the hop frequency (there is also a factor of 2^{23} that is removed by shifting).

    To simplify internal calculations, LOAD BWSCALE is in units of 200 Hz instead of 100 Hz used for all other LOADs. An alternative interpretation is that the scale factor loaded is one-half of the hopping bandwidth.

    The valid range for this value is 0 to 499500000 (1DC5C3E0h) corresponding to 0 Hz to 99.9 GHz hopping bandwidth. A scale factor of 0 means that all hops are at the same frequency which is equivalent to no hopping.

Example:

Operation	Hex Words (in order)	Remarks
LOAD BWSCALE 1.998 GHz	0306,6F70,0098	9990000 = 986F70h

Related Commands: LOAD BASE, LOAD HOP, LOAD OFFSET

Command:	LOAD DOPFACT <Doppler_factor>
Hex Value:	0310,XXXX$_{least}$,XXXX$_{most}$
Parameter:	Doppler factor [x2^{32}], positive value causes Doppler offset to be added
Valid Modes:	STOP

Description: This command loads the Doppler factor - which is multiplied by the nominal frequency to get the Doppler frequency offset. This offset is added to the nominal frequency (corresponding to the satellite approaching the ground terminal) if the Doppler factor is positive. Negative factors cause the Doppler offset to be subtracted from the nominal frequency.

The valid range is from -42950 to +42950. (Numbers outside this range will cause overflow in the intermediate calculations and result in erroneous frequencies.) The smallest step corresponds to 9 Hz Doppler offset on a 40 GHz nominal signal. When Doppler correction is not required, a Doppler factor of 0 should be used.

Example:

Operation	Hex Words (in order)	Remarks
LOAD DOPFACT 1.257E-8	0310,0036,0000	1.257E-8 x 2^{32} = 54 = 36h Corresponds to 503 Hz Doppler on 40 GHz

Related Commands: LOAD BASE, LOAD BWSCALE, LOAD OFFSET

60

Command: LOAD FCSPACE <bin_spacing>
Hex Value: 030A,XXXX$_{least}$,XXXX$_{most}$
Parameter: FSK tone bin spacing [100 Hz]
Valid Modes: STOP

Description: This command loads the FSK tone bin spacing. This is the frequency difference between adjacent bins. This value is multiplied by the FSK tone number (from the FSK/Channel interface) to get the FSK modulation frequency. The channel number is multiplied by eight times the FSK tone bin spacing (8-ary FSK) to get the channel offset. The modulation and channel offsets are then added to the hop frequency to produce the nominal frequency (before Doppler).

Valid range for bin spacing is 0 to 32677 (7FFFh). FSK modulation and channel selection are disabled by setting the bin spacing to 0.

Example:

Operation	Hex Words (in order)	Remarks
LOAD FCSPACE 40 kHz	030A,0190,0000	400 = 190h

Related Commands: LOAD FSKCHAN

Command: LOAD FLAGS <flag_value>
Hex Value: 031A,XXXX$_{least}$,XXXX$_{most}$
Parameter: Flag value [unitless] (32 bits of which only the lower 16 are used)
Valid Modes: STOP

Description: Loads the configuration control flags. The flag bits are as follows (with bit 0 as the LSB)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														Run	x2

x2:   Sciteq doubler flag - when set, the final synthesizer frequency is divided by two to account for a doubler placed after the Sciteq. A value of 0 for this flag or configuring for the Comstron synthesizer allows normal calculations.

Run: Powerup in RUN mode flag - when set, the HSC powers-up in RUN mode instead of STOP mode. Since the flags are loaded from the ROM on power-up, changing this flag has no effect.

Reserved: These bits are not used currently but should be programmed as all zeros to maintain compatibility with future enhancements.

Example:

Operation	Hex Words (in order)	Remarks
LOAD FLAGS 0	031A,0000,0000	No doubler No power-up run

Related Commands: LOAD LOSCI

Command: LOAD FSKCHAN <channel_bin>
Hex Value: 0308,XXXX$_{least}$,XXXX$_{most}$
Parameter: XXXXXXXX is the FSK Bin/Channel number (32 bits of which only the lower 8 bits are used). It is calculated using:

$$XXXXXXXX = 8N_{Channel} + N_{FSK\_Bin}$$

where: (25)
            XXXXXXXX is the parameter of the command
            $N_{Channel}$ is the channel number (0-31)
            N_{FSK_Bin} is the bin number (0-7)

Valid Modes: STOP (debugging)

Description: This command loads the FSK bin and channel number into the working area. Since this area is overwritten by the value read from the FSK/Channel interface before any calculations are done, there is no useful effect in using this command for other than debug purposes.

            Valid range is from 0 to 255 (FFh).

Example:

Operation	Hex Words (in order)	Remarks
LOAD FSKCHAN Channel 2, Bin 4	0308,0014,0000	20 = 14h

Related Commands: LOAD FCSPACE

63

LOAD HOP

Command:	LOAD HOP <hop_random_number>
Hex Value:	0304,XXXX$_{least}$,XXXX$_{most}$
Parameter:	Hop frequency random number [unitless]
Valid Modes:	STOP (debugging)

Description: This command loads the 24-bit random number to be used to compute the hop frequency, then causes the computations to be done, and finally sets the onboard latches with the new frequency. The frequency is subsequently loaded into the synthesizer on the next rising edge of the hop clock. If it is desired to load the synthesizer immediately, the CHANGE IMMEDIATE command should be used first.

A value of 0 sets the low edge of the hopping band, 8388608 (800000h) sets midband, while 16777215 (FFFFFFh) sets the high edge of the hopping band.

LOAD HOP is used during debugging to force the synthesizer to a certain area of the hopping bandwidth. Unlike LOAD LATCH which directly programs the onboard latches and is hence synthesizer dependent, LOAD HOP takes into account the type of synthesizer when the calculations are done.

Example:

Operation	Hex Words (in order)	Remarks
LOAD HOP	0304,0000,0080	8388608 = 800000h

Related Commands: CHANGE HOP, CHANGE IMMEDIATE, LOAD BASE, LOAD BWSCALE, LOAD LATCH, LOAD OFFSET

LOAD LATCH

Command:	LOAD LATCH <latch_value>
Hex Value:	0300,XXXX$_{least}$,XXXX$_{most}$
Parameter:	Latch value to be input to synthesizer
Valid Modes:	STOP (debugging)

Description: This command loads the parameter onto the on-board latches on the HSC which are the inputs to the hopping synthesizer. The format of the value is synthesizer dependent (see Appendix C for the formats). The synthesizer interface with the Comstron uses BCD values with the digits from 100 Hz to 8 GHz matching the least significant digit to the most significant digit of the latch value. Programming the Sciteq with this command is much more complicated because of the split-binary format. See Section 2.4.3 of the main document for more information on formatting for the Sciteq.

Once the on-board latches are loaded, the synthesizer will change frequency on the next rising edge of the hop clock. If the CHANGE IMMEDIATE command has been given, then the synthesizer will change immediately after the on-board latches are set with the LOAD LATCH command.

Alternatively, the LOAD HOP command is a synthesizer independent method of setting the synthesizer frequency but control is limited to the hopping range since the frequency computations are performed.

All values are accepted for this command but may not do anything meaningful depending on the type of synthesizer connected.

Example:

Operation	Hex Words (in order)	Remarks
LOAD LATCH 1.2345 GHz	0300,5000,1234	305418240 = 12345000h Comstron only

Related Commands: CHANGE HOP, CHANGE IMMEDIATE, LOAD HOP

LOAD LOCOM

Command:	LOAD LOCOM <Comstron_lo_freq>
Hex Value:	0312,XXXX$_{least}$,XXXX$_{most}$
Parameter:	Comstron LO frequency [100 Hz]
Valid Modes:	STOP

Description: This command loads the frequency of the Comstron LO for up or downconversion. This value is subtracted from the final transmit/receive frequency to get the desired synthesizer output frequency.

Because the different synthesizers have different output ranges, there will likely be a different conversion chain for each type of synthesizer. LOAD LOCOM is only used for the Comstron synthesizer - LOAD LOSCI is used for the Sciteq. This allows the user of the HSC to use the same program regardless of the synthesizer type currently connected. The synthesizer switch on the HSC controls whether LOCOM or LOSCI is used in the computations.

The valid range is 0 to 999000000 (3B8B87C0h) corresponding to 99.9 GHz.

Example:

Operation	Hex Words (in order)	Remarks
LOAD LOCOM 43 GHz	0312,4780,19A1	430000000 = 19A14780h

Related Commands: LOAD FLAGS, LOAD LOSCI

LOAD LOSCI

Command: LOAD LOSCI < Sciteq_lo_freq >
Hex Value: 0318,XXXX$_{least}$,XXXX$_{most}$
Parameter: Sciteq LO frequency [100 Hz]
Valid Modes: STOP

Description: This command loads the LO value for the Sciteq synthesizer that is used in the up or downconversion. This value is subtracted from the final transmit/receive frequency to get the desired synthesizer output frequency.

Since the Comstron and Sciteq may have different up/down conversion chains, the LO for the Sciteq can differ from that of the Comstron (which is loaded with LOAD LOCOM). The synthesizer switch on the HSC controls whether LOCOM or LOSCI is used in the computations.

See also LOAD FLAGS for the case where a doubler is used after the in the Sciteq synthesizer.

The valid range is 0 to 999000000 (3B8B87C0h) corresponding to 99.9 GHz.

Example:

Operation	Hex Words (in order)	Remarks
LOAD LOSCI 19.2 GHz	0318,B000,0B71	192000000 = B71B000h

Related Commands: LOAD FLAGS, LOAD LOCOM

67

Command: LOAD OFFSET <offset_freq>
Hex Value: 030E,XXXX$_{least}$,XXXX$_{most}$
Parameter: Guard band offset frequency [100 Hz]
Valid Modes: STOP

Description: This command loads the lower guard band offset. This value is added to the base frequency to produce the low edge of the hopping band without the effects of Doppler.

The valid range is 0 to 999000000 (3B8B87C0h) corresponding to 99.9 GHz.

Example:

Operation	Hex Words (in order)	Remarks
LOAD OFFSET 500 kHz	030E,1388,0000	5000 = 1388h

Related Commands: LOAD BASE, LOAD BWSCALE

Command: LOAD SEED <seed>
Hex Value: 0302,XXXX$_{least}$,XXXX$_{most}$
Parameter: Random number seed [unitless]
Valid Modes: STOP (debugging)

Description: This command loads a random number seed into the working area.
                Since the current initialization routine for the hop pattern overwrites this
                area with a seed of 1, this command has no useful effect except during
                debugging.

The valid range is 0 to 2147483647 (7FFFFFFFh).

Example:

Operation	Hex Words (in order)	Remarks
LOAD SEED 1	0302,0001,0000	1 = 1h

Related Commands: LOAD BWSCALE, LOAD HOP

Command: LOAD TIMELO <time_lower_32bits>
Hex Value: 0314,XXXX$_{lower,least}$,XXXX$_{lower,most}$
Parameter: Least significant 32 bits of 64 bit time [units of hops]
Valid Modes: STOP

Description: This command loads the least significant 32 bits of the time. Since most often the times used will not require the full 64 bits, this command also clears the upper 32 bits. Use LOAD TIMELO only to load a 32 bit time. Use LOAD TIMELO followed by LOAD TIMEHI for a 64 bit time. The time is specified as the number of hops since time 0.

This command should be used before DLSYNC, ULSYNC commands. If it is desired to have the HSC start running without synchronization, the LOAD TIMELO should be used before the RUN command. See Section 5 of Appendix A for start time restrictions.

Valid range for 64 bit time is 0 to 1.84467 x 10^{19} (FFFFFFFFFFFFFFFFh). The valid range for TIMELO parameter is 0 to 4294967295 (FFFFFFFFh).

Example:

Operation	Hex Words (in order)	Remarks
LOAD TIMELO 93675	0314,6DEB,0001	93675 = 16DEBh

Related Commands: LOAD TIMEHI

70

Command:	LOAD TIMEHI <time_higher_32bits>
Hex Value:	0316,XXXX$_{upper,least}$,XXXX$_{upper,most}$
Parameter:	Most significant 32 bits of 64 bit time [units of hops]
Valid Modes:	STOP

Description: This command loads the most significant 32 bits of the time and has no effect on the 32 least significant bits of the time. Since LOAD TIMELO automatically clears the upper 32 bits, LOAD TIMEHI should always follow (not precede) the LOAD TIMELO.

Use LOAD TIMELO only to load a 32 bit time. Use LOAD TIMELO followed by LOAD TIMEHI for a 64 bit time. The time is specified as the number of hops since time 0.

Time should be loaded before DLSYNC, ULSYNC commands. If it is desired to have the HSC start running without synchronization, the time should be loaded before the RUN command. See Section 5 of Appendix A for start time restrictions.

Valid range for 64 bit time is 0 to 1.84467×10^{19} (FFFFFFFFFFFFFFFFh). The valid range for TIMEHI parameter is 0 to 4294967295 (FFFFFFFFh).

Example:

Operation	Hex Words (in order)	Remarks
LOAD TIMELO 591751049 LOAD TIMEHI 1	0314,6789,2345 0316,0001,0000	4886718345 = 123456789h

Related Commands: LOAD TIMELO

READ

Command: READ <address>
Hex Value: 2XXX
Parameter: TMS320C25 address to be read
Valid Modes: STOP (debugging)

Description: The READ command is a special command that causes the HSC to output an internal RAM value serially through the Sync line. The lower 12 bits of the READ command specify the address to be output.

Once the command has been sent, the HSC prepares the value to be output on the Sync line, MSB first and LSB last. The bits are clocked by any command sent to the HSC (which is otherwise ignored) except the STOP command which finishes the READ. Repeating the READ command is a useful way of clocking. There are only 16 useful bits to be clocked, but no error is generated if more or fewer bits are clocked. The STOP command completes the READ and returns the HSC to STOP mode.

Because the Sync line is used for the READ command there can be frequency changes occurring in the synthesizer during the READ. As well, the Sync line is reset at the end of the READ so a previous CHANGE IMMEDIATE command is nullified. At the end of the command the HSC acts as if it had received a CHANGE HOP command.

Example:

Operation	Hex Words (in order)	Remarks
READ 30Ch	230C	Read base frequency at address 30Ch
READ 30Ch	230C	Bit 15 (MSB)
READ 30Ch	230C	Bit 14
...	...	...
READ 30Ch	230C	Bit 0 (LSB)
STOP	0	STOP mode

Related Commands: all LOADs

Command: RUN
Hex Value: 8000
Parameter: -
Valid Modes: DLSYNC, STOP, ULSYNC

Description: This command causes the HSC to switch to RUN mode where calculations are done each hop and the synthesizer frequency is updated at beginning of each hop. The time must have been loaded before the RUN command is invoked.

From STOP mode, this command is used to put the HSC in normal hopping operation.

From DLSYNC mode in the ground terminal, this command is used once the coarse acquisition verification has occurred on the second burst. The terminal then continues on with fine synchronization and tracking in RUN mode.

From ULSYNC mode in the ground terminal, this command is used once coarse uplink synchronization is achieved. The terminal then continues on with fine uplink synchronization and tracking in RUN mode.

Example:

Operation	Hex Words (in order)	Remarks
RUN	8000	

Related Commands: DLSYNC, STOP, ULSYNC

Command:	STOP
Hex Value:	0000
Parameter:	-
Valid Modes:	DLSYNC, RUN, ULSYNC

Description: This command switches the HSC to STOP mode. In this mode, the HSC only processes commands and the synthesizer is not updated. This command should be used three times for initialization. STOP mode is the only mode that allows the LOAD commands to be used.

From RUN mode, this command is used to stop hopping and change some parameters (using LOADs) before continuing on with normal hopping or synchronization.

From DLSYNC mode or ULSYNC mode, this command is used when synchronization has failed and it is necessary to adjust the time or other parameters before continuing.

The STOP command is also used to abort or complete a READ command in STOP mode.

Example:

Operation	Hex Words (in order)	Remarks
STOP	0000	

Related Commands: DLSYNC, READ, RUN, ULSYNC

Command: ULGO <burst_probe>
Hex Value: 02XX
Parameter:
        The probe number portion of the command (XX) is calculated using:

$$XX = 2\left(16N_{Burst} + N_{Probe}\right)$$
    where:
        XX is the least significant 8 bits of the command (0-3E)   (26)
        N_{Burst} is the burst number (0 for first 16 probes, 1 for the rest)
        N_{Probe} is the probe number (0-15) within the burst

Valid Modes: ULSYNC

Description: This command causes the sync probe to be transmitted in uplink synchronization mode. The switch to the new probe should occur in under 3 μs, so the command should be given exactly at the start of a hop or slightly before.

                This command should be given only after FCALC has been used to calculate all the probes for the current frame.

Example:

Operation	Hex Words (in order)	Remarks
ULGO Burst 1 Probe 3	0236	38 = 26h

Related Commands: DLGO, FCALC, ULSYNC

Command: ULSYNC <channel_bin>

Hex Value: 50XX

Parameter: XX is the FSK Bin/Channel number (8 bits). This is calculated using:

$$XX = 8N_{Channel} + N_{FSK\_Bin}$$

where:

    XX is the least significant 8 bits of the command (0-FF)   (27)

    $N_{Channel}$ is the channel number (0-31) for uplink sync probes

    N_{FSK_Bin} is the bin number (0-7) for uplink sync probes

Valid Modes: STOP

Description: This command causes the processor to switch from STOP mode to ULSYNC mode using the specified FSK tone and channel for the sync probes. It also computes the first two hops of the frame in preparation for subsequent FCALC commands.

The desired time should be set using LOAD TIMELO and if required LOAD TIMEHI prior to entering into ULSYNC mode. Section 5.2 of Appendix A details the method for determining the time to start.

The processor will set the Sync line when the transition the ULSYNC mode is complete. FCALC should then be used at the beginning of the frame followed by the appropriate ULGO commands during the sync probe burst time.

Uplink sync probes occur in bursts of 16 sequential probes.

Because of the speed needed to respond to ULGO commands, the processor's code has been streamlined to eliminate error checking. This means that illegal commands can cause the processor to enter an indeterminate state. Only RUN, STOP, FCALC or ULGO commands should be used in ULSYNC mode.

Example:

Operation	Hex Words (in order)	Remarks
ULSYNC Channel 2, Bin 3	5013	19 = 13h

Related Commands: DLSYNC, FCALC, RUN, STOP, ULGO

Appendix C: External Interfaces

1.   **Synchronization Controller Interface**

1.1   General

The Hopping Synthesizer Controller must be under command of the Synchronization Controller for initialization of parameters and for uplink and downlink synchronization. For synchronization, it is necessary for the HSC to precompute frequencies and switch them rapidly under control of the Synchronization Controller. The following sections deal with the interface between the Synchronization Controller and the HSC.

1.2   Hopping Synthesizer Controller Commands

The different types of commands for the HSC consist of one or three 16-bit words. The LOAD commands consists of a 16-bit opcode followed by 32 bits of data. All other commands are 16-bit. Some commands, such as READ or ULSYNC, include parameters in the 16 bits. The commands and their usage are fully detailed in Appendix B.

1.3   Hopping Synthesizer Controller Command Servicing

The normal sequence of issuing commands is to switch to STOP mode, perform all the necessary LOADS, switch to one of DLSYNC or ULSYNC modes and after synchronization is achieved, switch to RUN mode. The command servicing in each of these three stages is described below:

STOP Mode: In this mode, the HSC checks the interface for a command repeatedly regardless of the hop clock. Each command is fully decoded and invalid commands are ignored. Commands can be given as fast as the NReady line allows. Most commands are serviced in about 4 μs (except the READ command which is significantly longer).

DLSYNC/ULSYNC Mode: In these modes, the HSC checks the interface repeatedly for a command regardless of the hop clock. To improve performance in switching frequencies, there is no checking for invalid commands. Invalid commands could cause indeterminate results. The STOP, RUN, DLGO and ULGO commands are processed in approximately 3 μs. FCALC takes roughly 800 μs to complete calculations.

RUN Mode: In this mode, the HSC checks the interface only once every hop. Note that a slow hop clock will slow the command servicing as well. (This is most apparent when using the TESTHSC program described in Appendix F with external hop clock.) The only valid command is STOP and all others are ignored.

77

1.4   Hardware Interface

The interface consists of a 26-pin connector providing a 16-bit unidirectional parallel port with handshaking and a hop clock. The Sync line, normally used for handshaking, can be used by the READ command to return serial data from the HSC. The pinout of the connector is given in Fig. C1 and the timing diagrams are given in Fig. C2. The lines are detailed below:

D0-D15	The sixteen data lines originate from the Synchronization Controller and are active high TTL (transistor transistor logic) lines.
Gnd	Digital ground.
HClk	The HClk (hop clock) line originates from the Synchronization Controller and is an active high TTL line. This clock signal should change from 0 to 1 at the beginning of the hop period and have a pulse high time of at least 30 ns. In RUN mode, the pulse is used to latch the frequency synthesizer. See the Hop Clock portion of Fig. C2. for timing.
NReady	The NReady line originates from the HSC and is an active low TTL line. The NReady line is normally 0 indicating ready for a new data; when the new data is latched the NReady line is forced to 1 until the HSC has read the value from the latch. The latch will be read in less than one hop in RUN mode, and will be read in less than 3 μs in DLSYNC or ULSYNC modes. The old data will be lost if the HSC is not ready during the strobe of new data. Fig. C2. shows the timing relationship of the handshaking lines.
Strobe	The Strobe line originates from the Synchronization Controller and is an active high TTL line. The Strobe line should normally be 0 and, when new data is stable, go from 0 to 1 for at least 50 ns and then return to 0. The data lines must be stable before and after the transition and will be latched on the rising edge of Strobe. Fig. C2. shows the timing relationship of the handshaking lines.
Sync	The Sync line originates from the HSC and is an active high TTL line. This line is normally 0 and goes to 1 when the processor, having completed all calculations, is in the tight loop ready to respond immediately to DLGO (in DLSYNC mode) or ULGO (in ULSYNC mode) commands. The timing for a switch to DLSYNC mode is shown in Fig. C2.
	The Sync line is a buffered version of the C25's XF output and also controls the latching of the synthesizer (which latches immediately in synchronization modes and on the hop clock in other modes).
	The Synchronization Controller should check this line to determine when the HSC is ready to change frequencies rapidly. This line returns to 0 when RUN or STOP is commanded.

The Sync line is also used during a READ operation to provide the serial data out with the order being from MSB to LSB. In this case it is clocked by sending a command (any command, other than STOP, is ignored except for clocking). The new bit is valid when the Ready line goes back up after a command was sent. The STOP command (all zeros) indicates the end of the read and the Sync line then returned to 0. (An implied CHANGE HOP command.)

```
      Gnd Sync HClk Gnd D14 D12 D10 D8 D6 D4 D2 D0 Gnd
       25   23   21  19  17  15  13  11  9  7  5  3  1
      ┌──────────────────────────────────────────────┐
      │  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠        │
      │  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠  ⊠        │
      └──────────────────────────────────────────────┘
       26   24   22  20  18  16  14  12  10  8  6  4  2
      Gnd Ready Strobe Gnd D15 D13 D11 D9 D7 D5 D3 D1 Gnd
```
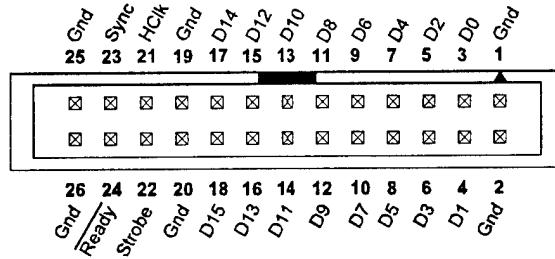
Fig. C1. Synchronization Controller interface connector (P2) pinout.

2.   **FSK/Channel Interface**

Note that the FSK/Channel interface cannot be left unconnected. If FSK/Channel selection is not desired, then all lines should be set to 0. See Section 2.3 of this appendix for more details on a stub to accomplish this.

2.1   General

The FSK/Channel Interface allows FSK data and channel selection to be input to the HSC. The FSK modulation and channel selection for each hop are sampled by the HSC soon after the rising edge of the hop clock. An additional feature of this interface is the RFoff control line that can cause the Sciteq synthesizer to turn off the RF power (by applying 30 dB of attenuation). The Comstron synthesizer doesn't have any control over the RF power, so the turning off of power is simulated by an offset of 1 GHz to the normal hopping pattern effectively switching the synthesizer to a value outside of the band of interest. There should then be no appreciable signal remaining after the dehopped IF (intermediate frequency) filter in the receiver.

2.2   Hardware Interface

The interface consists of a 10-pin connector providing 3 bits for FSK tone specification, 5 bits for channel number and 1 bit for RF power control. There is no handshaking; all 9 lines are sampled near the rising edge of the hop clock for the next hop (ex: on the rising edge of the hop clock of hop #1, the 9 lines for hop #2 are sampled). The hop clock is derived internally or externally from the Synchronization Controller Interface (see Section 1 of this appendix) depending on the configuration switch. The pinout for the FSK/Channel interface can be found in Fig. C3. The descriptions of the FSK/Channel interface lines are given below.

Command Strobe

D0-D15 ⟩⟨ Command #1 ⟩⟨ Command #2 ⟩⟨ Command #3 ⟩⟨

Strobe →| 50⁺ns |←— 10⁺ns |←→|

Ready |←——1 hop max——→|

Hop Clock

Hop Number | Hop #N-1 | Hop #N | Hop #N+1 |

HClk |←—30⁺ns—→|

Changing to DLSYNC Mode

D0-D15 ⟨ DLSYNC Command ⟩ ⟨ DLGO B0 H1 ⟩

Strobe

→| 2 µs max |←—

Ready All frequencies are calculated

Sync

Synth Output | DL Burst 0, Hop 0 | DL Burst 0, Hop1 |

● ● ●
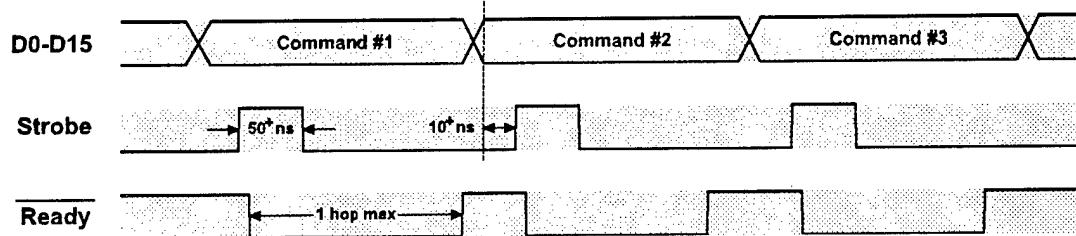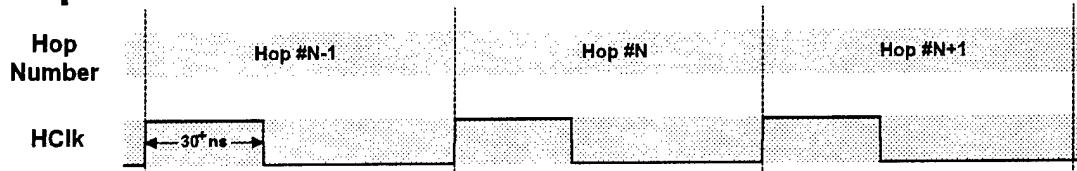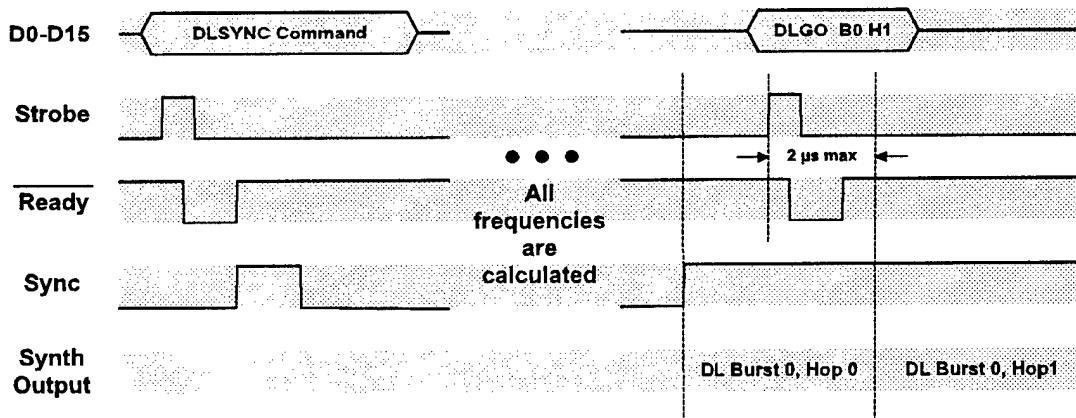
Fig. C2. Hopping Synthesizer Controller interface timing diagram.

FSK0-FSK2   The three active high TTL lines originate from the formatted data source (Frame Formatter) and are sampled on the rising edge of the hop clock, 1 hop early. They indicate the 8-ary FSK symbol with FSK0 as the LSB and FSK2 as the MSB. The symbols range from 0 (000 binary) to 7 (111 binary) where 0 is the lowest frequency tone. When no FSK modulation is desired, all three lines should be tied to ground. Note that the amount of modulation is affected by the FCSPACE parameter - when FCSPACE is set to zero there is no modulation despite changes to the FSK lines.

80

Chan0-Chan4	These five lines originate from the Frame Formatter and are sampled (along with FSK0-FSK2) on the previous hop's rising edge of the hop clock. The five bits indicate one of the 32 possible channels and are active high TTL. Chan0 is the LSB and Chan4 is the MSB. The channel numbers range from 0 (00000 binary) to 31 (11111 binary) where 0 is the lowest frequency channel. When no channel selection is desired, all five lines should be tied to ground. Note that the channel spacing is eight times the FCSPACE parameter - when FCSPACE is set to zero there is no channel selection despite changes on the Chan lines.
RFoff	This active high TTL line comes from the Frame Formatter and, when set, causes the RF power of the synthesizer to be turned off (Sciteq) or a 1 GHz offset to be applied to the nominal frequency (Comstron). RFoff is an active high TTL line which is sampled on the previous rising edge of the hop clock. This bit can be used to disable transmission during non-allocated times or it can be used to simulate jamming. This pin should be tied to ground when not in use.
Gnd	Digital ground.
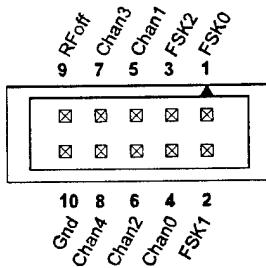
Fig. C3. FSK/Channel interface connector (P3) pinout.

2.3   Stub

In the event that FSK modulation and channel selection are not used, and no control is needed over the RF power, all of the pins on the interface should be tied to ground. (Alternatively, the RFoff pin can be tied to ground with the FCSPACE parameter set to 0.) It is necessary to tie unused lines to ground because TTL inputs will go to 1 if left open - resulting in FSK tone of 7, channel 31 and RF off.

The simplest way to build a stub is to have a short piece of ribbon cable on a female plug where all of the 10 lines of the ribbon cable are soldered together. This results in FSK tone of 0 (no offset), channel 0 (no offset) and RF on.

3.   **Frequency Synthesizer Interface**

3.1   General

The HSC drives the Frequency Synthesizer through a 50-pin parallel port. The interface is set up for direct crimped ribbon cable connection to a Comstron synthesizer and can be connected to the Sciteq with a custom cable. The output for the Comstron is in BCD form with only eight decades controlled, the rest are forced to zero. For the Sciteq, the output is a split-range binary form with the finest resolution bits forced to zero.

3.2   Comstron Synthesizer

The Comstron synthesizer and its interface requirements are documented in [4] and [5]. The synthesizer accepts a BCD frequency selection up to 18 GHz (on some models) with a resolution of 0.1 Hz. The models used in testing the board could cover from 0 to 4 GHz with rapid switching in less than 1 μs.

The cable connecting the HSC to the Comstron synthesizer is a 50 pin ribbon cable with a header crimped on the controller end and a 50 pin D connector connected on the synthesizer end.

Note the cable connecting the Comstron to the Hopping Synthesizer Controller should be no longer that 3 feet. This is detailed in [4] and [5].

The interface consists of a 50 pin connector made up of numerous frequency selection lines with one strobe line and some unused power lines. The pinout of the connector is shown in Fig. C4. The pinout by function is listed in Tables C1 and C2. The crimp D-connector at the Comstron follows a different pin numbering convention and the mapping of this connector to the board connector is given in Table C3. The description of the interface pins are given below:

0.1 Hz (1) to 10 GHz (1)   These 45 data lines originate from the HSC and are active high TTL lines. The Comstron loads these lines with four standard TTL loads. They are latched by the synthesizer on the rising edge of the Strobe line. They should be stable before this edge. Note that the 0.1 Hz, 1 Hz, 10 Hz and 10 GHz decades are tied to zero on the board and cannot be commanded.

The most significant bit of the 1 GHz decade (pin 24) and the least significant bit of the 10 GHz decade (pin 10) are both driven through 220 Ω resistors because of incompatibilities between two models of Comstron. One model places +5V on pin 24 while it is an input on the other. Pin 10 is labelled TBD (assume the worst) for one model and is an input for the other. These resistors enable the interface to work with either model. Larger resistors coupled with the capacitance of the interface cable may cause excessive time delays.

82

Strobe	This active high TTL line originates in the HSC; the rising edge causes the synthesizer to latch the frequency selection lines. The rising edge occurs 50 ns after the frequency selection is valid and stays high for 200 ns.
+5V	This is a five volt source from the Comstron. This line is unconnected on the HSC board.
Logic Ground	Tied to digital ground on the board.
Chassis Ground	Tied to digital ground on the board.
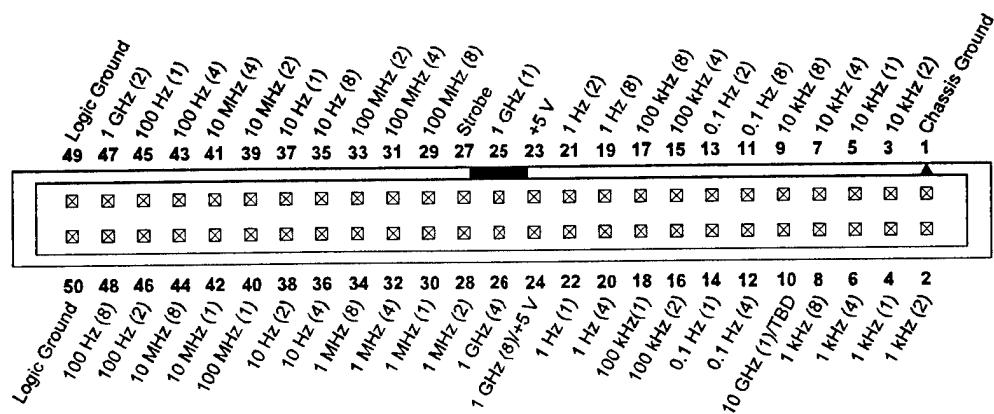
Fig. C4. Synthesizer interface connector (P1) pinout for Comstron.

Top row (odd pins):
49 Logic Ground, 47 1 GHz (2), 45 100 Hz (1), 43 100 Hz (4), 41 10 MHz (4), 39 10 MHz (2), 37 10 Hz (1), 35 10 Hz (8), 33 100 MHz (2), 31 100 MHz (4), 29 100 MHz (8), 27 Strobe, 25 1 GHz (1), 23 +5V, 21 1 Hz (2), 19 1 Hz (8), 17 100 kHz (8), 15 100 kHz (4), 13 0.1 Hz (2), 11 0.1 Hz (8), 9 10 kHz (8), 7 10 kHz (4), 5 10 kHz (1), 3 10 kHz (2), 1 Chassis Ground

Bottom row (even pins):
50 Logic Ground, 48 100 Hz (8), 46 100 Hz (2), 44 10 MHz (8), 42 10 MHz (1), 40 100 MHz (1), 38 10 Hz (2), 36 10 Hz (4), 34 1 MHz (8), 32 1 MHz (4), 30 1 MHz (1), 28 1 MHz (2), 26 1 GHz (4), 24 1 GHz (8)/+5 V, 22 1 Hz (1), 20 1 Hz (4), 18 100 kHz (1), 16 100 kHz (2), 14 0.1 Hz (1), 12 0.1 Hz (4), 10 10 GHz (1)/TBD, 8 1 kHz (8), 6 1 kHz (4), 4 1 kHz (1), 2 1 kHz (2)

Table C1. Functional pinout for frequency selection on P1 for Comstron.

Decade	Bit value			
	8	4	2	1
10 GHz*	-	-	-	10
1 GHz	24	26	47	25
100 MHz	29	31	33	40
10 MHz	44	41	39	42
1 MHz	34	32	28	30
100 kHz	17	15	16	18
10 kHz	9	7	3	5
1 kHz	8	6	2	4
100 Hz	48	43	46	45
10 Hz*	35	36	38	37
1 Hz*	19	20	21	22
0.1 Hz*	11	12	13	14

* these decades are tied to zero

Table C2. Functional pinout for non-frequency selection on P1 for Comstron.

Function	Pin Number(s)
Strobe	27
+5V	23
Logic Ground	49, 50
Chassis Ground	1

Table C3. Mapping of HSC P1 connector to Comstron connector

HSC 50 pin Header	Comstron 50 pin D	HSC 50 pin Header	Comstron 50 pin D
1	1	26	38
2	26	27	14
3	2	28	39
4	27	29	15
5	3	30	40
6	28	31	16
7	4	32	41
8	29	33	17
9	5	34	42
10	30	35	18
11	6	36	43
12	31	37	19
13	7	38	44
14	32	39	20
15	8	40	45
16	33	41	21
17	9	42	46
18	34	43	22
19	10	44	47
20	35	45	23
21	11	46	48
22	36	47	24
23	12	48	49
24	37	49	25
25	13	50	50

The difference in pin numbers between the two connectors in this table is due to the non-standard pin numbers of the D-connector when it is crimped to a ribbon cable.

3.3   Sciteq Synthesizer

The details of the Sciteq synthesizer interfacing are documented in [6]. The synthesizer accepts a split-range binary frequency selection from 1 to 2 GHz with a resolution slightly larger than 65 mHz. The Sciteq also has a phase control connector which can be used for phase modulation. The phase control connector is not used or affected by the HSC.

The cable connecting the HSC to the Sciteq synthesizer is a 50 pin cable with a header on the controller end and a 50 pin sub-miniature D connector (3 rows of pins) connected on the synthesizer end. This cable should be as short as possible.

The interface consists of 50 pins including fine and coarse frequency selection lines with control lines and some unused lines. The pinout of the connector is shown in Fig. C5. The pinout by function is listed in Table C4. The mapping of the cable is given in Table C5. For more information on how to format the fine and coarse frequency selection bits, see Section 2.4.3 in the main document. The description of the interface pins are given below:

F29 - F34	These 6 coarse frequency select lines originate from the HSC and are active high TTL lines. The synthesizer follows these lines when the Strobe line is high and latches them on the falling edge. The frequency select lines should be stable before the leading edge of the Strobe.
	The most significant coarse bit (F34) corresponds to 800 MHz and the least significant coarse bit (F29) corresponds to 25 MHz. The value set by the coarse bits must be in the range of 0 - 975 MHz.
F00 - F28	These 28 fine frequency select lines originate from the HSC and are active high TTL lines. The synthesizer follows these lines when the Strobe line is high and latches them on the falling edge. The frequency select lines should be stable before the leading edge of the Strobe. Note that F00 to F08 are tied to ground.
	The most significant fine bit (F28) corresponds to 17.5 MHz and the least significant fine bit (F00) corresponds to roughly 65 mHz ($17500000*2^{-28}$ Hz). The least significant non-grounded bit is F09 corresponding roughly to 33 Hz. The value set by the fine bits must be in the range of 5 - 29.99999999 MHz.
Strobe	This active high TTL line originates in the HSC; when high, the synthesizer follows the frequency selection lines and latches them on the falling edge. The rising edge occurs 50 ns after the frequency selection is valid and stays high for 200 ns.
Reset(RF off)	This active high TTL line originates from the HSC. When high, the RF output is attenuated by 30 dB. This can be used when it is not desired to transmit.
Ground	Tied to digital ground on the board.

Top row pin labels (odd pins):

Ground · N/C · F09 · F11 · F31 · F30 · F05 · F08 · F34 · N/C · N/C · Strobe · Reset (RF off) · N/C · F02 · F04 · F24 · F23 · N/C · F00 · F20 · F19 · F17 · F18 · Ground

49 47 45 43 41 39 37 35 33 31 29 27 25 23 21 19 17 15 13 11 9 7 5 3 1

Bottom row pin labels (even pins):

50 48 46 44 42 40 38 36 34 32 30 28 26 24 22 20 18 16 14 12 10 8 6 4 2

Ground · F12 · F10 · F32 · F29 · F33 · F06 · F07 · F28 · F27 · F25 · F26 · N/C · N/C · F01 · F03 · F21 · F22 · N/C · N/C · N/C · F16 · F15 · F13 · F14

Fig. C5. Synthesizer interface connector (P1) pinout for Sciteq.

Table C4. Frequency selection interface connector pin numbers.

Function	Pin Number(s)
Coarse Frequency F34 - F29	33,40,44,41,39,42
Fine Frequency F28 - F09	34,32,38,30,17,15,16,18,9,7,3,5,8,6,2,4,48,43,46,45
Fine Frequency* F08 - F00	35,36,38,37,19,20,21,22,11,12,13,14
Reset (RF off)	25
Strobe	27
Ground	1,49,50

* these bits are tied to zero

87

Table C5. Mapping of HSC P1 connector to Sciteq connector

HSC P1 50 pin Header	Sciteq 50 pin D	HSC P1 50 pin Header	Sciteq 50 pin D
1	-	26	-
2	39	27	46
3	8	28	43
4	23	29	-
5	40	30	27
6	7	31	-
7	25	32	11
8	24	33	30
9	41	34	28
10	-	35	37
11	2	36	21
12	-	37	36
13	-	38	4
14	-	39	12
15	42	40	13
16	26	41	29
17	10	42	44
18	9	43	38
19	20	44	45
20	3	45	5
21	35	46	22
22	19	47	-
23	-	48	6
24	-	49,50	1,17,18,
25	32		33,34,50

4. Serial Port Interface

This interface is currently unused, but may be used in future versions.

4.1 General

Originally, when the HSC was first conceived, it was desirable that the hopping pattern be generated externally by a dedicated pseudo-random pattern generator. (Delays in the development of the generator caused a simplified random pattern to be incorporated into the HSC for test purposes.) The serial Port Interface is been incorporated to allow later revisions of the software to accept an external pseudo-random sequence to be used to derive the hopping pattern.

88

4.2   Hardware Interface

This interface consists of buffered versions of the C25's built-in serial port. The programming, timing and configuration of this port are described in [7]. The pinout is shown in Fig. C6. and the pin descriptions follow:

DR   The Data Receive line is active high TTL and originates from the pattern generator. It will be used to pass the hopping sequence and may be used for other information flowing from the generator to the HSC.

DX   The Data Transmit line is active high TTL and originates from the HSC. It will be used to pass the time of interest to the generator and may be used for other information flowing to the generator.

CLKR   The falling edge of the Receive Clock line is used to latch the DR line. This TTL line originates from the generator. (If transmit and receive clocking are at the same rate, it may be tied to the CLKX line inside the generator.)

CLKX   The rising edge of the Transmit Clock line is used to change the DX data. The falling edge of the Transmit Clock should be used by the generator to latch the DX data. This TTL line originates from the HSC. The clock rate is set to 400 kHz.

FSR   The falling edge of the Receive Frame Synchronization pulse initiates the data reception. This TTL line originates from the generator.

FSX   The falling edge of the Transmit Frame Synchronization pulse initiates the data transmission. This TTL line originates from the HSC.

Gnd   Digital Ground
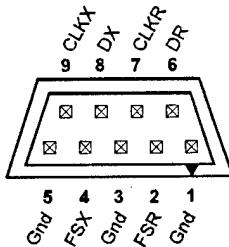
Fig. C6. Serial Port interface connector (P4) pinout.

89

Appendix D: Hardware Details

1.   General

This appendix contains the schematics, component list and layout of the Hopping Synthesizer Controller. The prototype board and its differences are also documented in this appendix. Details on the EPLD programming and the firmware can be found in Appendix E.

There are several connectors used in the HSC. The connectors on the rear panel (P1 to P4) correspond exactly to the associated connector on the printed circuit board inside the box (J1 to J4).

2.   Schematics

The schematics for the HSC are given in two sheets Fig. D1 and Fig. D2. Since 1991, when these schematics were made, there has been only one change to the board to reduce the glitches due to ground bounce. The ground bounce problem is detailed in Section 4.2 of the main document. The two changes to the circuits are:

a. Addition of three 62 pF capacitors on the latch strobe lines. Specifically they were added to U2 (pin 17 to pin 19), U3 (pin 11 to pin 10) and U7 (pin 11 to pin 10).

b. Substitution of four 74LS374 for 74F374 in positions U3, U4, U7 and U8.
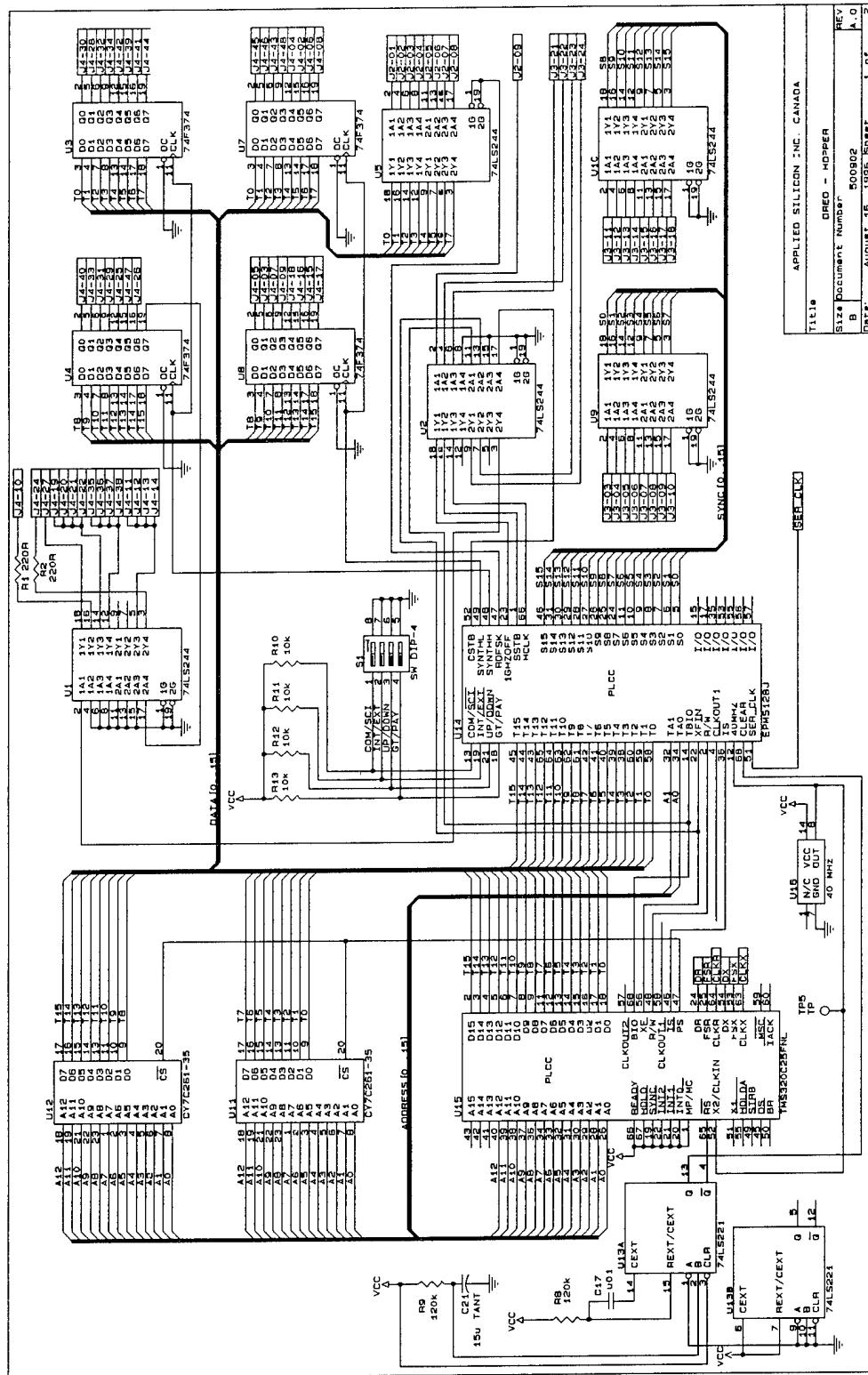
Fig. D1. Hopping Synthesizer Controller board schematic (sheet 1 of 2).

92
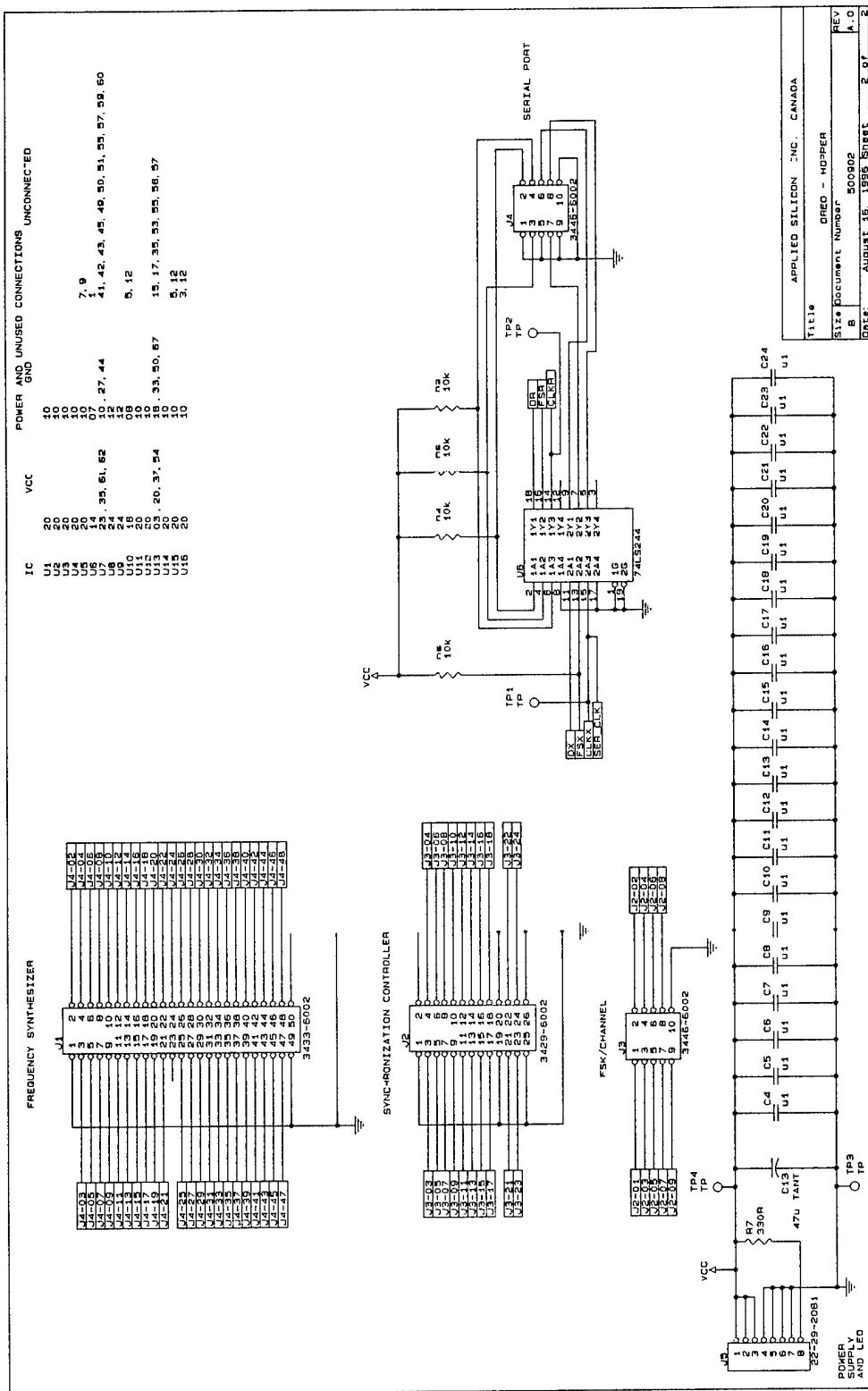
Fig. D2. Hopping Synthesizer Controller board schematic (sheet 2 of 2).

93

3.   Layout Description

The printed circuit board layout of the HSC is shown in Fig. D3. All parts are standard off-the-shelf with the exception of the EPLD (U14) and the EPROMS (U11, U12). The EPLD containing the interface logic and the EPROMS containing the digital signal processor (DSP) program must be programmed. There are unused IC positions on the bottom of the board (Spare 1 and Spare 2) to allow for extra RAM, extra EPROMs or additional TTL. As well, there is unused capacity in the EPLD (U14) which could be used without modifying the board layout.
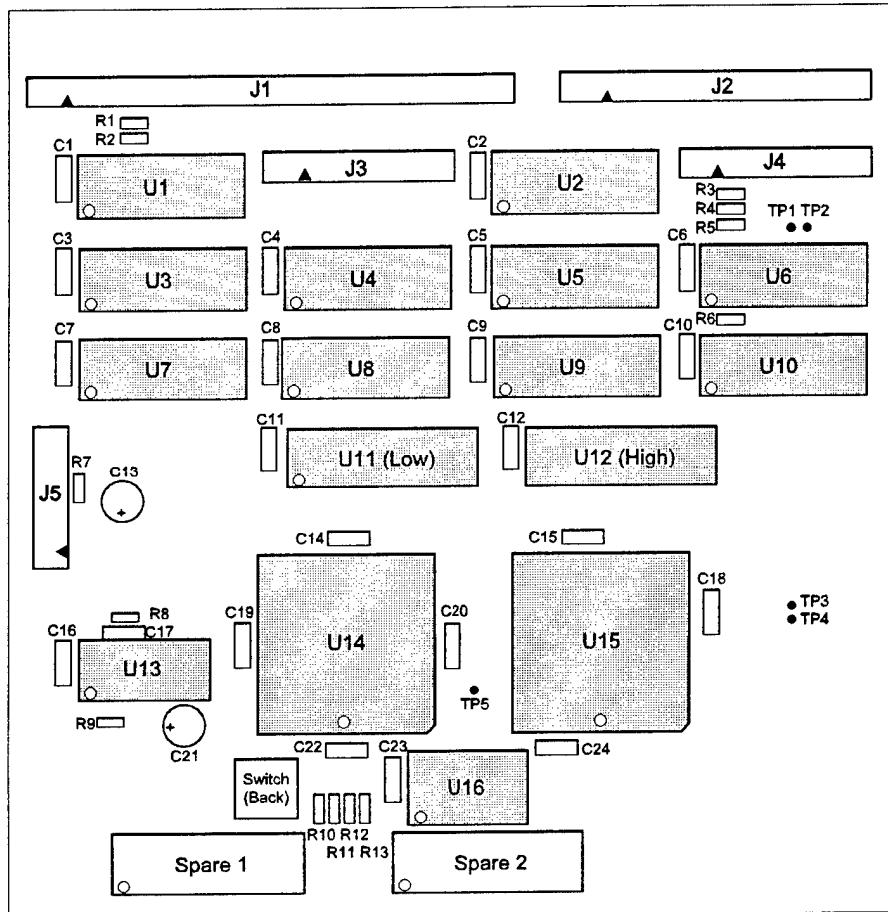
Fig. D3. Hopping Synthesizer Controller printed circuit board layout.

Fig. D4. shows the front panel layout of the HSC with the DC (direct current) power indicating LED (light emitting diode) and the AC (alternating current) switch.
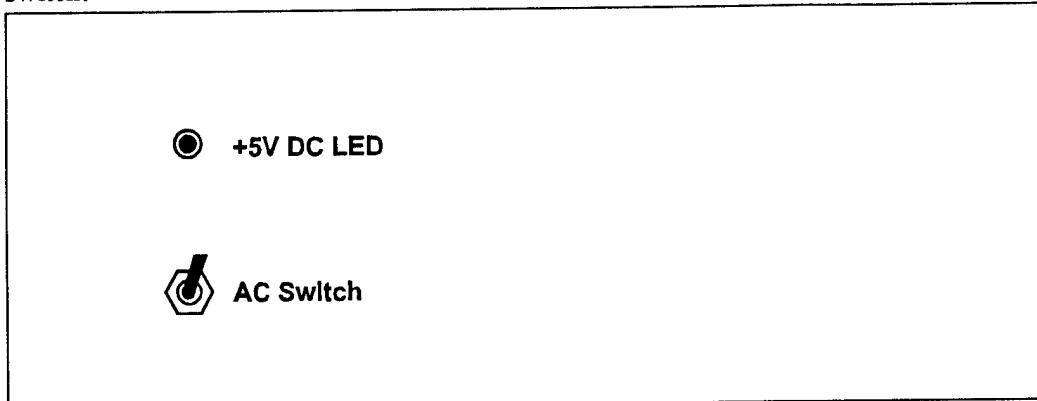
Fig. D4. Hopping Synthesizer Controller front panel layout.

The rear panel layout is shown in Fig. D5. The AC socket is on the upper right with a 1 A fuse below. The four interface connectors (P1 to P4) are described in Appendix C. Note that FSK/Channel connector (P3) requires a grounding stub to tie all inputs to 0's when FSK/Channel selection is not used.
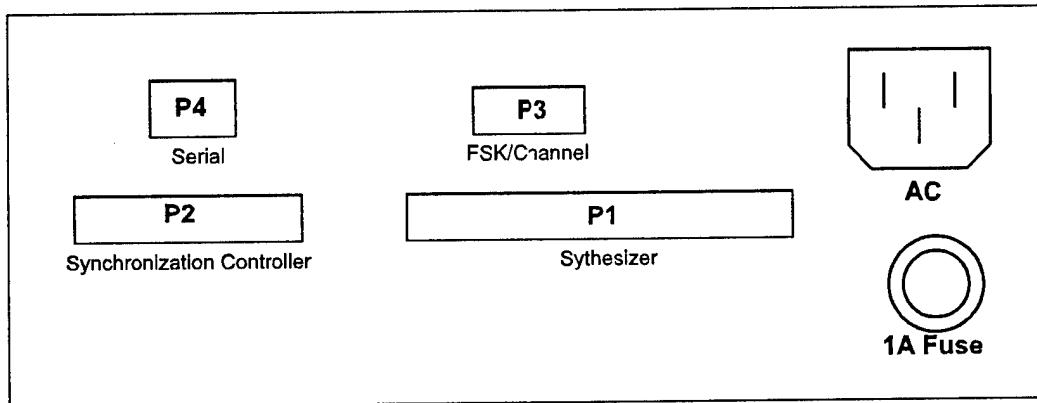
Fig. D5. Hopping Synthesizer Controller rear panel layout.

The bottom panel layout is shown in Fig. D6. This panel provides access to the configuration switches for the HSC. The setting of these switches is further detailed in Section 2 of Appendix A.
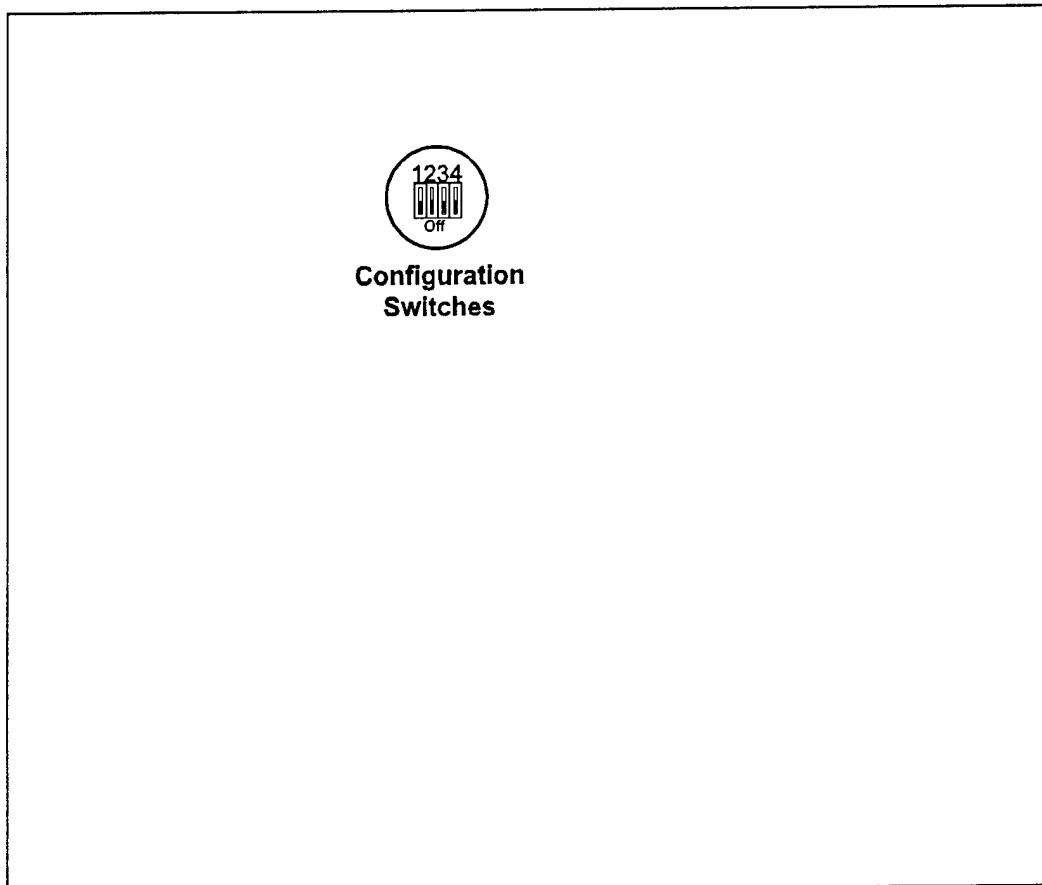
Fig. D6. Hopping Synthesizer Controller bottom panel layout.

4.   **Key Component List**

4.1   Integrated Circuits

U1,U2,U5,U6,U9,U10	74LS244	Octal line drivers
U3,U4,U7,U8	74LS374	Octal D-flipflops
U11,U12	7C261	Cypress 8k x 8 EPROM, 35 ns
U13	74LS221	Dual monostable multivibrators
U14	EPM5128JC-2	Altera 128 macro-EPLD, 30 ns
U15	TMS320C25FNL	TI Digital Signal Processor
U16	F1100E	Fox 40 MHz crystal oscillator

4.2   Discrete Components

R1,R2,	220 Ω, ¼ W
R3,R4,R5,R6	10 kΩ, ¼ W
R7	330 Ω, ¼ W
R8,R9	120 kΩ, ¼ W
R10,R11,R12,R13	10 kΩ, ¼ W

C1-C12	0.1 μF
C13	47 μF, tantalum
C14-C16	0.1 μF
C17	0.01 μF
C18-C20	0.1 μF
C21	15 μF, tantalum
C22-C24	0.1 μF
C25-C27	62 pF (not shown on schematic or layout)

TP1-TP5	Test point post
Switch	DIP switch, single throw, 4 wide
Spare 1, Spare 2	These are unused socket positions available for expansion

4.3   Other

J1	50 pin connector, double row, 0.1" spacing
J2	26 pin connector, double row, 0.1" spacing
J3,J4	10 pin connector, double row, 0.1" spacing
J5	8 pin connector, single row, 0.1" spacing
Power Supply	LSS-34-5, 5 V at 2 A

5. Prototype Board

5.1 General

The prototype board is a speedwire board developed prior to the manufacturing of the HSC. Because it is a prototype, some connections are different and not all features are incorporated. The differences are presented in the next section, followed by the details of the special PLCC adaptor used on the prototype board. The prototype board layout is shown in Fig. D7.
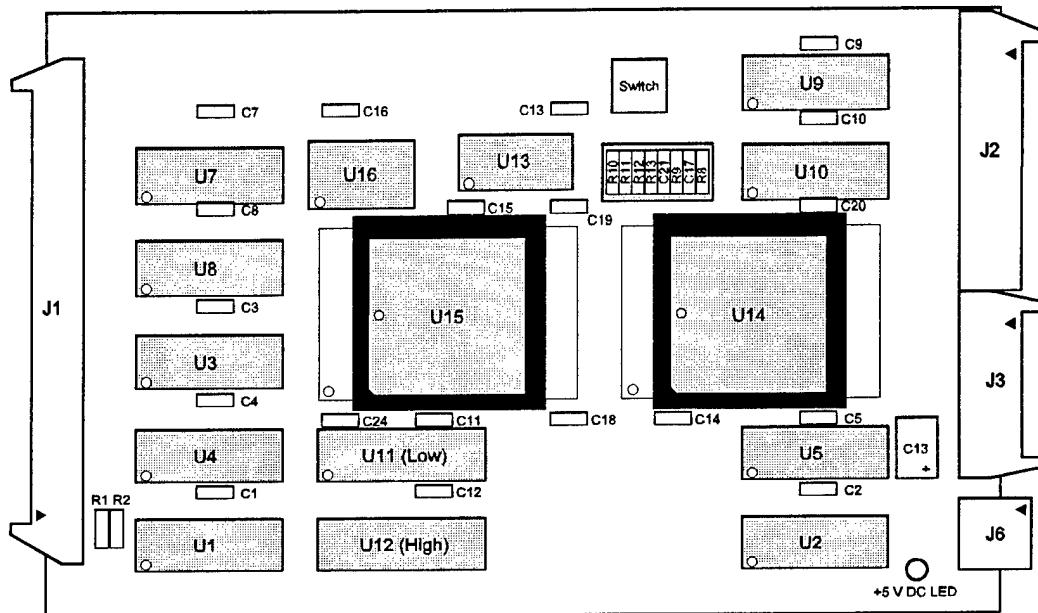
Fig. D7. Prototype board layout.

5.2 Differences between the HSC and the Prototype Board

a. The serial interface is not part of the prototype. The EPLD used is an earlier version that does not include the serial interface. The J4 connector (serial) and the associated buffers U6 are not included.

b. The power supply is not part of the prototype. There is no J5 connector, instead there is a J6 connector with the pinout shown in Fig. D8.
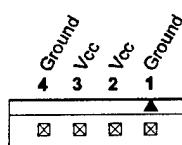
Fig. D8. Prototype board power connector (J6).

c. There are no testpoints on the prototype board.

d. The PLCC integrated circuits use a special adaptor on the prototype boards. This is detailed in the next section.

5.3   PLCC Adaptor used on Prototype Board

On the prototype board, U14 and U15 reside in 68 pin PLCC to 0.1" x 0.3" adaptors. The speedwire board does not support PLCC packages because the board has rows 0.3" apart with pins at 0.1" spacing. The adaptor converts the PLCC pinout to the 0.1" x 0.3" pinout. Since this spacing is used mostly for DIPs, this adaptor will be referred to as a PLCC to DIP adaptor (despite the fact that there are more than 2 rows of pins). A diagram of the adaptor is given in Fig. D9. The relationship of PLCC pin numbers and DIP pin numbers is given in Tables D1 and D2. Note that the pin numbers used in the schematic are the PLCC pin numbers.
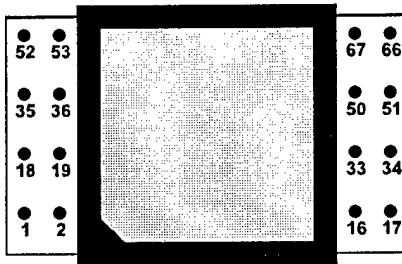
Fig. D9. Top view of PLCC-to-DIP adaptor.

Table D1. DIP to PLCC pin number mapping.

DIP	PLCC	DIP	PLCC	DIP	PLCC	DIP	PLCC
1	7	18	2	35	1	52	63
2	8	19	3	36	65	53	64
3	9	20	4	37	66	54	60
4	11	21	5	38	67	55	61
5	10	22	6	39	68	56	62
6	12	23	13	40	58	57	59
7	14	24	15	41	56	58	57
8	16	25	17	42	54	59	55
9	18	26	19	43	52	60	53
10	20	27	21	44	50	61	51
11	22	28	23	45	48	62	49
12	24	29	25	46	46	63	47
13	26	30	32	47	38	64	45
14	28	31	31	48	37	65	43
15	27	32	34	49	40	66	44
16	30	33	33	50	39	67	42
17	29	34	36	51	35	68	41

99

Table D2. PLCC to DIP pin number mapping.

PLCC	DIP	PLCC	DIP	PLCC	DIP	PLCC	DIP
1	35	18	9	35	51	52	43
2	18	19	26	36	34	53	60
3	19	20	10	37	48	54	42
4	20	21	27	38	47	55	59
5	21	22	11	39	50	56	41
6	22	23	28	40	49	57	58
7	1	24	12	41	68	58	40
8	2	25	29	42	67	59	57
9	3	26	13	43	65	60	54
10	5	27	15	44	66	61	55
11	4	28	14	45	64	62	56
12	6	29	17	46	46	63	52
13	23	30	16	47	63	64	53
14	7	31	31	48	45	65	36
15	24	32	30	49	62	66	37
16	8	33	33	50	44	67	38
17	25	34	32	51	61	68	39

Appendix E: Firmware/Software Details

1.   General

This appendix details the firmware/software used in the HSC. It consists of the logic firmware residing in an EPLD and the DSP program residing in two EPROMS. The EPLD was configured with a schematic compiler whereas the DSP program was implemented in TMS320C25 assembly language.

2.   EPLD Schematics

The EPLD, Altera 5128, is an Erasable Programmable Logic Device that replaces a collection of TTL logic. This is programmed using the MAX+Plus software from Altera. The EPLD schematic, Fig. E1, includes several TTL macros taken from the MAX+Plus TTL Macro Library. Some of the macros in the EPLD schematic are custom macros whose schematics are given in Fig. E2-E5. These custom macros are:

DIV2	Frequency divide by 2
DIV25	Frequency divide by 5 and 25
SHIFT2	2-bit D-flipflop shift register
81MUXB	8 wide 2:1 multiplexer, bus form
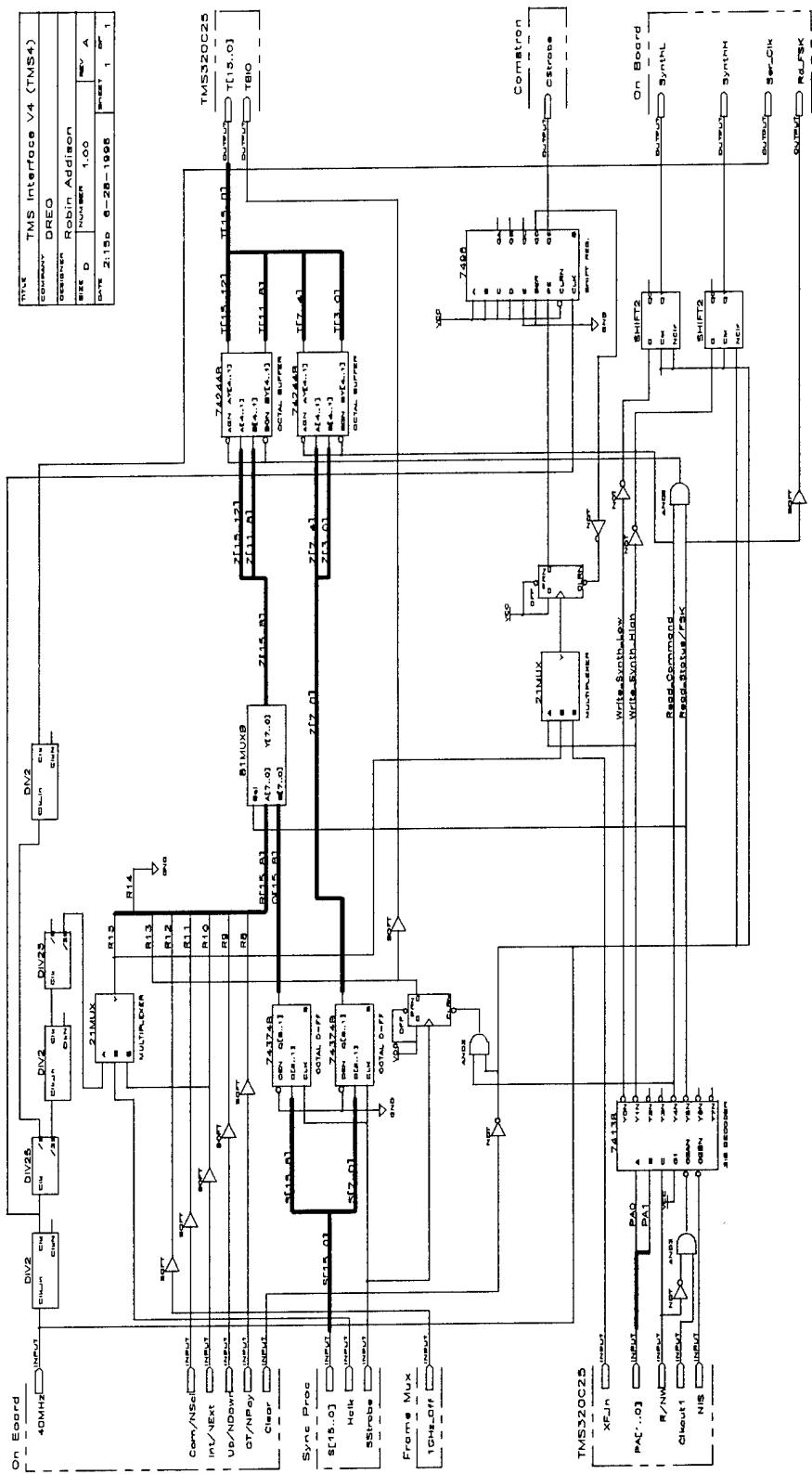
Fig. E1. TMS4 interface EPLD diagram.

TITLE /2 Divider (DIV2)

COMPANY DREO/ED/CNS

DESIGNER Robin Addison

SIZE B | NUMBER 1.00 | REV A

DATE 3:32p 4-19-1996 | SHEET 1 OF 1

OUTPUT Clk

OUTPUT ClkN

NOT

FREQDIV

G DV2 DV4 DV8 DV16 CLR CLK

FREQ. DIVIDER

VCC

GND

Clk_In INPUT
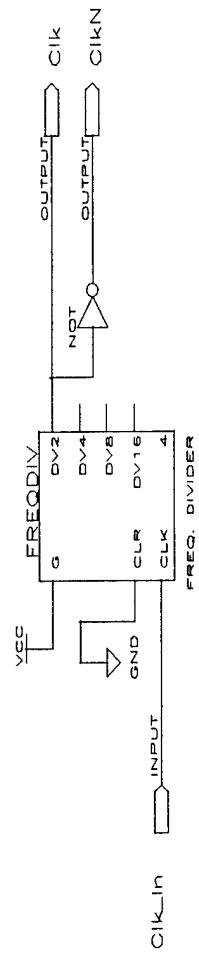
Fig. E2. DIV2 EPLD macro diagram.
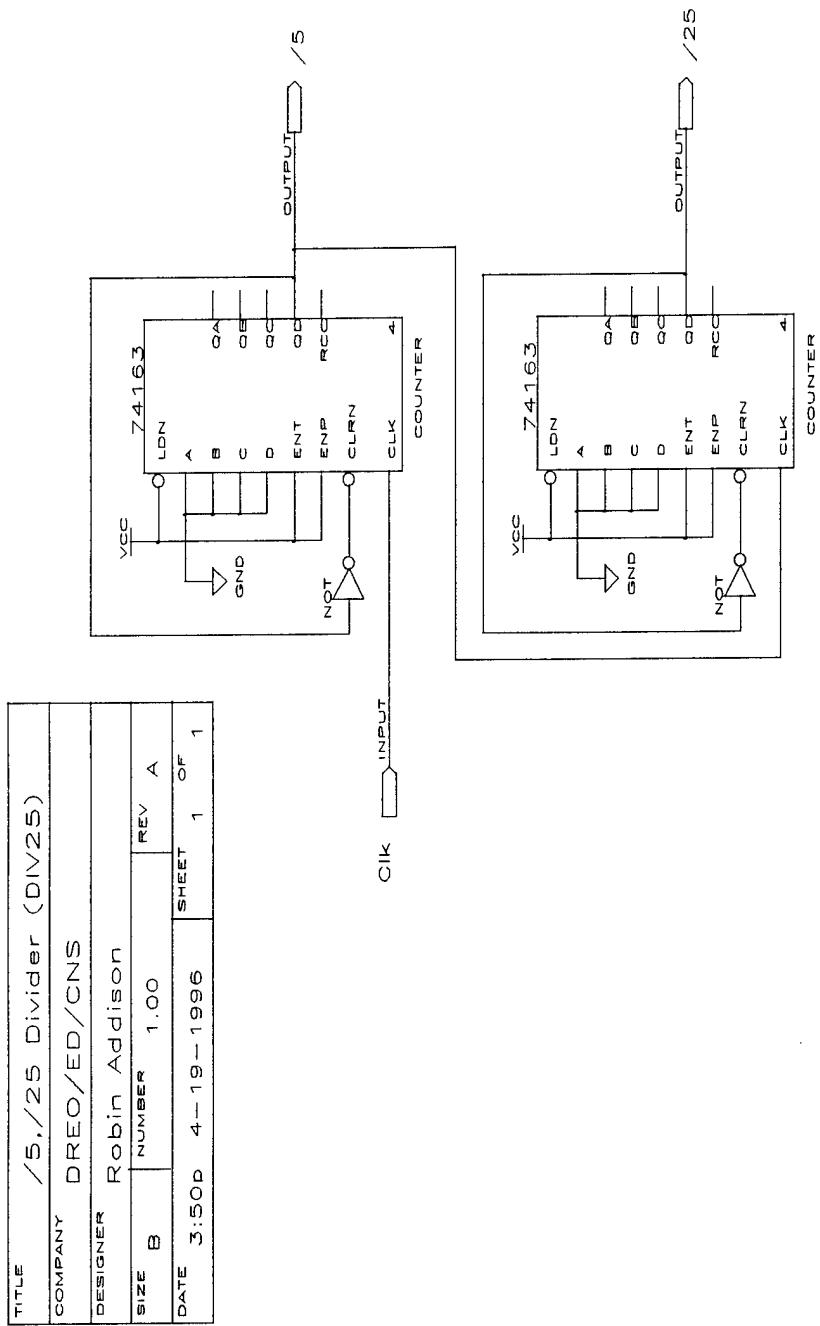
Fig. E3. DIV25 EPLD macro diagram.

104
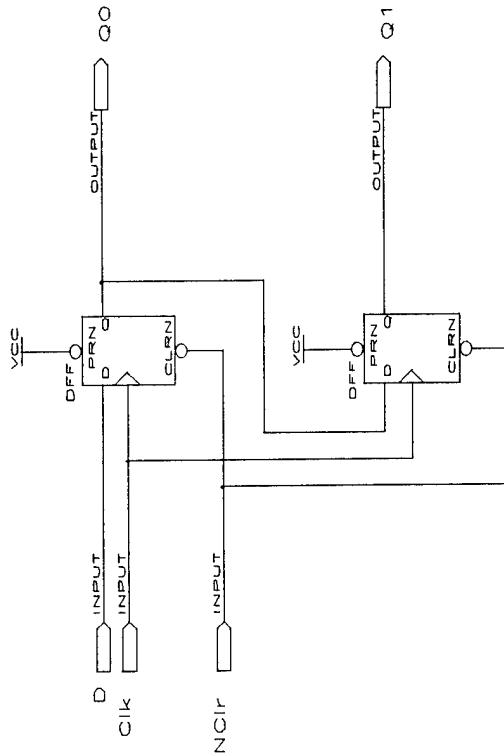
Fig. E4. SHIFT2 EPLD macro diagram.
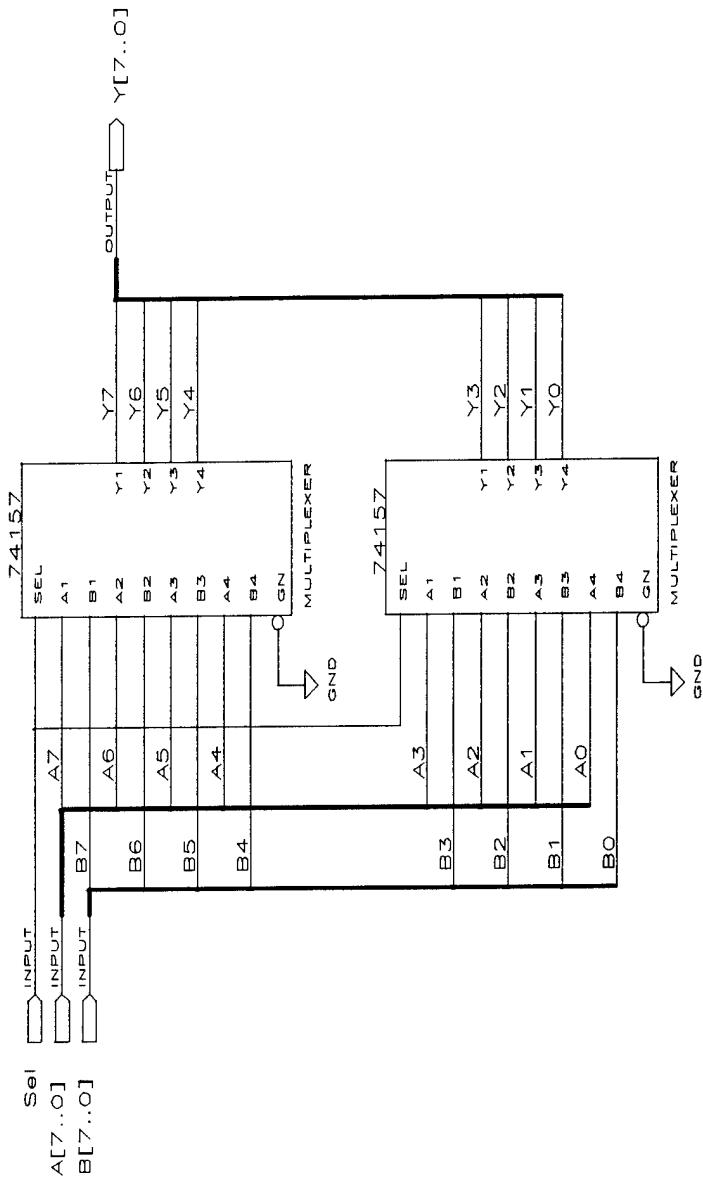
Fig. E5. 81MUXB EPLD macro diagram.

3. Software Listings

3.1 General

The software was assembled and linked on a Dell System 433E running under DOS 5.00. The Texas Instruments TMS320C1x/TMS320C2x Assembly Language Tools were used for assembly, linking and reformatting. The make file used to generate the EPROM files follows:

```
HSC.HI:          HSC.OUT
                 DSPROM HSC.OUT -t
                 DSPROM HSC.OUT -i

HSC.OUT:         HSC.OBJ HSC_CALC.OBJ HSC_RAN.OBJ
                 DSPLNK HSC.CMD

HSC.OBJ:         HSC.ASM
                 DSPA HSC.ASM -l

HSC_CALC.OBJ:    HSC_CALC.ASM
                 DSPA HSC_CALC.ASM -l

HSC_RAN.OBJ:  HSC_RAN.ASM
                 DSPA HSC_RAN.ASM -l
```

The command file used by the linker (HSC.CMD) is given below:

```
HSC.OBJ               /* Main program                      */
HSC_CALC.OBJ          /* PN -> BCD Frequency calculation   */
HSC_RAN.OBJ           /* Random number generation          */

-e HSC                /* Entry point                       */
-o HSC.OUT            /* Executable file                   */
-m HSC.MAP            /* Map file                          */

MEMORY
{
            PAGE 0:   VECTORS:    origin =    0H, length =   03FH
                      PROG:       origin = 1000H, length = 0EF00H
            PAGE 1:   DATA:       origin =  300H, length = 0FD00H
}
SECTIONS
{
            IRUPTS: { } > VECTORS    PAGE 0
            .text:  { } > PROG       PAGE 0
            .data:  { } > PROG       PAGE 0
            .bss:   { }              PAGE 1
}
```

The EPROM files were then transferred one at a time (high and low byte data) to a Data I/O 29B programmer using PROMLink from Data I/O. Once the file is transferred, the Cypress CY7C261-35WC (8k x 8 EPROM) is programmed using a

107

family/pinout code of EF31.

The next three sections are the software listings for the main program (HSC), the calculation routines (HSC_CALC) and the random number generation routines (HSC_RAN).

```
MULGOL:   .set    023EH           ; Go to last uplink sync hop
MULGOX:   .set    0242H           ; Go to 2nd hop after last uplink sync hop
MFCALC:   .set    5800H           ; Calculate a frames worth of hops (plus 2)


; Load Commands (Stop mode)
MLDLTC:   .set    0300H           ; Load latches directly
MLDSED:   .set    0302H           ; Load random number seed (not used)
MLDHOP:   .set    0304H           ; Load hop number, calc freq, load latches
MLDBWS:   .set    0306H           ; Load BW scale (HBW/2 or units 200 Hz)
MLDFSK:   .set    0308H           ; Load FSK/channel (not used)
MLDFCS:   .set    030AH           ; Load FSK/channel spacing (100 Hz)
MLDBAS:   .set    030CH           ; Load base frequency (100 Hz)
MLDOFF:   .set    030EH           ; Load offset frequency - guard band (100 Hz)
MLDDPF:   .set    0310H           ; Load doppler factor (x2^32, neg subtacts dop)
MLDLOC:   .set    0312H           ; Load Comstron LO frequency - (100 Hz)
MLDTLO:   .set    0314H           ; Load time low - and clear Time High (hops)
MLDTHI:   .set    0316H           ; Load time high (2^32 hops)
MLDLOS:   .set    0318H           ; Load Sciteq LO frequency - (100 Hz)
MLDFLG:   .set    031AH           ; Load divide by 2 flags


; Stop Mode commands (See also Load Commands)
MDEFLT:   .set    0400H           ; Reload default values
MCGHOP:   .set    0500H           ; Change frequency on hop clock (normal)
MCGIMM:   .set    0501H           ; Change frequency immediately (sync)
MREAD:    .set    2000H           ; Read internal variables
MRDEND:   .set    2FFFH           ; - end of valid read commands


CPRETY:   .set    1111H
```

```
;==============================================================================
;
;          Variable area (BSS)
;
;------------------------------------------------------------------------------
;
          .bss    ANSWER,2        ; Calculated value to be sent to latches
          .bss    SEED,2          ; Seed for random number

          .bss    HOPNUM,2        ; Lower 24 bits of seed for hop number
          .bss    BWSCAL,2        ; BW scale factor [200 Hz]

          .bss    FSKCHN,2        ; FSK/Channel value
          .bss    FCSPAC,2        ; FSK/Channel spacing [100 Hz]

          .bss    FBASE,2         ; Base frequency [100 Hz]
          .bss    FOFFST,2        ; Guard band offset frequency [100 Hz]

          .bss    DOPFCT,2        ; Doppler factor [x2^32]
          .bss    FLOCOM,2        ; Comstron LO frequency [100 Hz]

          .bss    TIMELO,2        ; Time (lower 32 bits) [hops]
          .bss    TIMEHI,2        ; Time (upper 32 bits)

          .bss    FLOSCI,2        ; Sciteq LO frequency [100 Hz]
          .bss    FLAGS,2         ; Configuration flags

          .bss    RAN_A,1         ; Constant for random number generation
          .bss    ZERO,1          ; Constant 0
          .bss    ONE,1           ; Constant 1
          .bss    COFFFF,1        ; Constant FFFFh
          .bss    C01000,1        ; Constant 1000h
          .bss    CULBAS,1        ; Constant equal to lowest ULSYNC command

          .bss    X0,1            ; X value for 32-bit multiplication (lower)
          .bss    X1,1            ; (upper)
          .bss    Y0,1            ; Y value for 32-bit multiplication (lower)
          .bss    Y1,1            ; (upper)
```

110

```
        .bss    Z0,1        ; Z 64-bit answer for 32-multiplication (low 16)
        .bss    Z1,1        ; (mid-lo 16 bits)
        .bss    Z2,1        ; (mid-hi 16 bits)
        .bss    Z3,1        ; (upper 16 bits)
        .bss    SIGN,1      ; Sign for 32-bit multiplication
        .bss    TMP,1       ; Temporary location

        .bss    CMD,1       ; Command received from controller
        .bss    DATA1,1     ; Data value associated with command (lower)
        .bss    DATA2,1     ; (upper)

        .bss    WPNT,1      ; Pointer to command/data area (CMD&DATA1/2)
        .bss    STAT,1      ; Status register value
        .bss    MODE,1      ; Mode of operation RUN/STOP/ULSYNC/DLSYNC

        .bss    HTAB,8      ; Table used for BCD conversion (long const)
        .bss    HEND,1
        .bss    LTAB,4      ; Table used for BCD conversion (short const)
        .bss    LEND,1
```

```
;==============================================================================
;
;       Reset and Interrupt Vector Area
;
;------------------------------------------------------------------------------
;

        .sect   "IRUPTS"
RESET:  B       MAIN
INT0:   B       BADINT
INT1:   B       BADINT
INT2:   B       BADINT
        .space  16 * 16
TINT:   B       BADINT
RINT:   B       BADINT
XINT:   B       BADINT
USER:   B       BADINT
```

```
;==============================================================================
;
;       BADINT - Illegal interrupt handler
;
;------------------------------------------------------------------------------
;

        .text
BADINT:                     ; Illegal interrupt - set the XF flag and hang
        SXF
BADLOP: B       BADLOP
```

```
;==============================================================================
;
;       MAIN - Main program
;
;------------------------------------------------------------------------------
;

MAIN:
        DINT            ; 1     ; Ensure interrupts disabled
        ROVM            ; 1     ; Reset overflow mask
        LARP    AR1     ; 1     ; Set ARP to AR0
        LDPK    0       ; 1     ; Point to page 0 (control area)
        LACK    0       ; 1     ; Mask off all interrupts
        SACL    4       ; 1
        RSXM            ; 1     ; Disable sign extension
        SPM     0       ; 1     ; No shift on P register
```

111

```
            LDPK      HOPNUM      ; 1         ; Point to data area

            IN        STAT,STATIN ; 2         ; Load in status for config info
            ZALS      STAT
            ANDK      0FFH
            SACL      FSKCHN
            CALL      BCDINI      ; 25        ; Initialize tables for BCD convert
            CALL      RANINI      ; 19        ; Initialize random number gen
            CALL      SETDEF      ; 116       ; Set defaults for calculations
            LALK      MULSYN
            SACL      CULBAS
            LALK      01000H
            SACL      C01000

            IN        CMD,CMDIN   ; 2         ; Ensure command latch is cleared
            LRLK      ARO,CMD     ; 3         ; No commands received yet
            SAR       ARO,WPNT
            SXF                   ; 1         ; Latch synth immediately
            CALL      PRETTY      ; 4.7 s
            RXF                   ; 1         ; Latch synth on hop clock

            BIT       FLAGS,FRUN  ; Check for startup in RUN mode
            BBNZ      BEGRUN

BEGSTP:     LRLK      ARO,MSTOP   ; 3         ; Start in STOP mode
            SAR       ARO,MODE
            B         STOP

BEGRUN:     LRLK      ARO,MRUN    ; 3         ; Start in RUN mode
            SAR       ARO,MODE
            B         RUN

STOP:                             ; Stop Mode - no hopping
            CALL      INSTR       ; -         ; Check if commanded, change modes
            BLZ       STRUN
            BNZ       SYNC
            B         STOP
STRUN:      CALL      RANJMP      ; 21+29/    ; Get random # for given time
            B         RUN

RUN:                              ; Run Mode - normal hopping
            CALL      WAIT        ; -         ; Wait until leading edge of hop
            CALL      LDHOPN      ; 11
            CALL      CALC        ; 386
            OUT       ANSWER,SYNTHL          ; Output to synth, wait for hop to latch
            OUT       ANSWER+1,SYNTHH
            CALL      RAN         ; 24        ; Set up Random # for next hop
            CALL      INCTIM      ; 19
            CALL      INSTR       ; -         ; Check if commanded, change modes
            BZ        STOP
            BGZ       SYNC
            B         RUN

SYNC:                             ; Sync Mode - fast response hops
            ZALS      MODE        ; Subtract of ULSYNC base forFSK/Chan
            SUB       CULBAS                 ;   FSK/Channel value
            BLZ       DLSYNC                 ;   - downlink sync comd less than base

ULSYNC:                           ; Uplink Sync Mode
            SACL      FSKCHN                 ;     - save uplink FSK/Channel
            CALL      RANJMP      ; 21+29/    ; Jump to desired frame start
            CALL      RAN         ; 24        ; Move to hop #2 (of 0-N)
            CALL      INCTIM      ; 19
            CALL      RAN         ; 24
            CALL      INCTIM      ; 19
            SXF                   ; Indicate ready & immediate latch

ULOOP:                            ; Wait for command
```

112

```
          BIOZ      ULOOP
          IN        CMD,CMDIN
          ZALH      CMD
          BZ        USTOP                    ; Respond to Stop Mode command
          BLZ       URUN                     ; Respond to Run Mode command
          LARP      AR1
          LAR       AR1,CMD                           ; Respond to Go commands
          LRLK      ARO,MFCALC               ; Check for frame calc command
          CMPR      0
          BBNZ      PRECAL
          OUT       *+,SYNTHL                ; Otherwise, assume uplink go
          OUT       *,SYNTHH
          B         ULOOP
PRECAL:                                      ; Calculate all sync hop for a frame
          LRLK      AR2,288-2-1              ;  (note that we start 2 hop into frame
                                             ;             and last FDATA is at AR2=0)
FDATA:                                       ; Skip the data hops
          CALL      RAN          ; 24
          CALL      INCTIM       ; 19
          LARP      AR2
          BANZ      FDATA,*-
          LRLK      AR2,MULGOO
FSYNC:                                       ; Calculate and save the sync hops
          CALL      LDHOPN       ; 11
          CALL      CALC         ; 386
          LARP      AR2
          SACL      *+
          SACH      *+
          CALL      RAN          ; 24
          CALL      INCTIM       ; 19
          LARP      AR2
          LRLK      ARO,MULGOX+2
          CMPR      0
          BBZ       FSYNC
          B         ULOOP


USTOP:    RXF                                ; Set to hop clock latch mode
          SACH      MODE
          B         STOP
URUN:     RXF                                ; Set to hop clock latch mode
          SACH      MODE
          LARP      AR2                      ; Set up second hop of next frame
          LRLK      AR2,MULGOX               ; into latches, so it is valid at the
          OUT       *+,SYNTHL                ; next clock edge
          OUT       *,SYNTHH
          B         RUN


DLSYNC:                                      ; Downlink Sync Mode
          CALL      RANJMP       ; 21+29/    ; Calc 1st donwlink sync hop (T+0)
          CALL      LDHOPN       ; 11
          CALL      CALC         ; 386
          LARP      AR1
          LRLK      AR1,MDLGOO
          SACL      *+
          SACH      *+

          CALL      RAN          ; 24        ; Calc 2nd donwlink sync hop (T+2)
          CALL      INCTIM       ; 19
          CALL      RAN          ; 24
          CALL      INCTIM       ; 19
          CALL      LDHOPN       ; 11
          CALL      CALC         ; 386
          LARP      AR1
          LRLK      AR1,MDLGO1
          SACL      *+
          SACH      *+
```

113

```
        CALL    RAN         ; 24     ; Calc 3rd donwlink sync hop (T+4)
        CALL    INCTIM      ; 19
        CALL    RAN         ; 24
        CALL    INCTIM      ; 19
        CALL    LDHOPN      ; 11
        CALL    CALC        ; 386
        LARP    AR1
        LRLK    AR1,MDLGO2
        SACL    *+
        SACH    *+


        CALL    RAN         ; 24     ; Calc 4th donwlink sync hop (T+6)
        CALL    INCTIM      ; 19
        CALL    RAN         ; 24
        CALL    INCTIM      ; 19
        CALL    LDHOPN      ; 11
        CALL    CALC        ; 386
        LARP    AR1
        LRLK    AR1,MDLGO3
        SACL    *+
        SACH    *+


        CALL    STPTIM      ; 10     ; Calc 5th donwlink sync hop (T+5120)
        CALL    RANJMP      ; 21+29/
        CALL    LDHOPN      ; 11
        CALL    CALC        ; 386
        LARP    AR1
        LRLK    AR1,MDLGO4
        SACL    *+
        SACH    *+


        CALL    RAN         ; 24     ; Set random # and time to hop after
        CALL    INCTIM      ; 19     ; 5th sync hop - so can RUN immediately

        SXF                          ; Set to immediate latch mode
        LARP    AR1
        LRLK    AR1,MDLGO0
        OUT     *+,SYNTHL            ; Output first hop to synth
        OUT     *,SYNTHH

SLOOP:                               ; Wait for command
        BIOZ    SLOOP
        IN      CMD,CMDIN
        ZALH    CMD
        BZ      SSTOP                ; Respond to Stop Mode command
        BLZ     SRUN                 ; Respond to Run Mode command
        LAR     AR1,CMD                      ; Respond to Go commands
        OUT     *+,SYNTHL
        OUT     *,SYNTHH
        B       SLOOP
SSTOP:  RXF                          ; Set to hop clock latch mode
        SACH    MODE
        B       STOP
SRUN:   RXF                          ; Set to hop clock latch mode
        SACH    MODE
        B       RUN
```

```
;==============================================================================
;
;         INSTR - check for and process command
;
;------------------------------------------------------------------------------
;
;               in:    -
;               out:   STAT,FSKCHN
;               used:  ACC,ARO,AR1,CMD,DATA1,DATA2,WPNT,XF(flag)
;
```

114

```
;--------------------------------------------------------------------------

INSTR:
        BIOZ    IDONE                   ; Input latch not full?
        LARP    AR1                     ; Get command
        LAR     AR1,WPNT
        IN      *+,CMDIN
        SAR     AR1,WPNT
        LRLK    ARO,DATA1+1             ; Cmd + 1 data ?
        CMPR    0
        BBNZ    IDONE
        LRLK    ARO,DATA2+1             ; Cmd + 2 data ?
        CMPR    0
        BBNZ    LOADS
        LAR     AR1,CMD                 ; Run mode command?
        LRLK    ARO,MRUN
        CMPR    0
        BBNZ    CMODE
        LRLK    ARO,MDLSYN              ; Donwlink Sync mode command?
        CMPR    0
        BBNZ    CMODE
        LRLK    ARO,MSTOP               ; Stop mode command?
        CMPR    0
        BBNZ    CMODE
        LRLK    ARO,MDEFLT              ; Defaults command?
        CMPR    0
        BBNZ    CSETDF
        LRLK    ARO,MCGHOP              ; Change frequency on hop command?
        CMPR    0
        BBNZ    CRXF
        LRLK    ARO,MCGIMM              ; Change frequency immediately command?
        CMPR    0
        BBNZ    CSXF
        LRLK    ARO,MFCALC              ; Frame Calc (uplink sync) command ?
        CMPR    0
        BBNZ    CGOS
        LRLK    ARO,MDLGOO              ; Valid downlink go command ?
        CMPR    1                       ;   (GO 0 to GO 4)
        BBNZ    ERROR
        LRLK    ARO,MDLGO4+1
        CMPR    1
        BBNZ    CGOS
        LRLK    ARO,MULGOO              ; Valid uplink go command ?
        CMPR    1                       ;   (ULGO 0 to ULGO 31)
        BBNZ    ERROR
        LRLK    ARO,MULGOL+1
        CMPR    1
        BBNZ    CGOS
        LRLK    ARO,MLDLTC              ; Valid load command ?
        CMPR    1                       ;   (LOAD LATCH to LOAD FLAGS)
        BBNZ    ERROR
        LRLK    ARO,MLDFLG+1
        CMPR    1
        BBNZ    CLOAD
        LRLK    ARO,MREAD
        CMPR    1
        BBNZ    ERROR
        LRLK    ARO,MRDEND
        CMPR    1
        BBNZ    READS
        LRLK    ARO,MULSYN              ; Valid uplink sync command ?
        CMPR    1                       ;   (ULSYNC 0 to ULSYNC 255)
        BBNZ    ERROR
        LRLK    ARO,MULEND+1
        CMPR    1
        BBNZ    CMODE
        B       ERROR                   ; Unrecognized command
```

115

```
READS:   ZALS    CMD                         ; Read values through XF
         ANDK    0FFFH                       ;  MSB to LSB
         SACL    CMD                         ;  1 bit / command
         LAR     AR1,CMD                     ;       STOP to finish
         ZALH    *
         RXF
RDWAIT:  BIOZ    RDWAIT
         ROL
         BC      RDSET
RDCLR:   RXF
         B       RDNEXT
RDSET:   SXF
RDNEXT:  IN      CMD,CMDIN
         LAR     AR1,CMD
         BANZ    RDWAIT,*-
         RXF
         B       INEXT

LOADS:   LAR     AR1,CMD                          ; Load commands
         ZALS    DATA1
         SACL    *+
         ZALS    DATA2
         SACL    *-
         LRLK    AR0,MLDLTC                  ; If LOAD answer, output to latches
         CMPR    0
         BBZ     ISLDSD
         OUT     ANSWER,SYNTHL               ; Output to synth, wait for hop to latch
         OUT     ANSWER+1,SYNTHH
         B       INEXT
ISLDSD:  LRLK    AR0,MLDHOP                  ; If LOAD HOP, calc frequency then out
         CMPR    0
         BBZ     ISLDTL
         CALL    CALC            ; 386
         OUT     ANSWER,SYNTHL
         OUT     ANSWER+1,SYNTHH
         B       INEXT
ISLDTL:  LRLK    AR0,MLDTLO                  ; If LOAD TIMELO, zero TIMEHI
         CMPR    0
         BBZ     INEXT
         LRLK    AR1,MLDTHI
         SACH    *+
         SACH    *-
         B       INEXT

CMODE:   LAR     AR1,CMD                     ; Change mode commands
         SAR     AR1,MODE                    ; Save mode
         B       INEXT

CGOS:    B       INEXT                       ; Go, frame calc commands, ignored

CLOAD:   B       IDONE                       ; Load commands, held until all data
                                             ;  is present

CSETDF:  CALL    SETDEF          ; 116           ; Set default command
         B       INEXT

CRXF:    RXF                                 ; Set to latch on hop clock
         B       INEXT

CSXF:    SXF                                 ; Set to latch immediately
         B       INEXT

ERROR:   B       INEXT                       ; Errors, unrecognized command

INEXT:   LRLK    AR1,CMD                     ; Reset input pointer to accept the
         SAR     AR1,WPNT                    ;  next command
         B       IDONE
```

116

```
IDONE:     ZALH    MODE                ; Load the next (or current) mode
           RET                         ;   and return


;===============================================================================
;
;          WAIT - wait for upswing of hop clock
;
;-------------------------------------------------------------------------------
;
;                   in:   -
;                   out:  STAT,FSKCHN
;                   used: ACC
;
;                   cycles:
;                         N/A            (includes call/return)
;                   time:
;                         N/A            (includes call/return)
;
;-------------------------------------------------------------------------------

WAIT:
           IN      STAT,STATIN         ; Ensure clock is low
           ZALH    STAT
           BLZ     WAIT
WAITUP:
           IN      STAT,STATIN         ; Wait for low to high transition
           ZALH    STAT
           BGZ     WAITUP
           ZALS    STAT                ; Save the FSK/Channel bits
           ANDK    0FFH
           SACL    FSKCHN

           RET


;===============================================================================
;
;          INCTIM - Advance time by one hop
;
;-------------------------------------------------------------------------------
;
;                   in:   -
;                   out:  SEED,TIMELO
;                   used: ACC,AR0,AR1
;
;                   cycles:
;                         19             (includes call/return)
;                   time:
;                         1.9 us         (includes call/return)
;
;-------------------------------------------------------------------------------

INCTIM:
           LARP    AR1           ; 1 (16)
           LAR     AR1,TIMELO    ; 1
           ADRK    1             ; 1
           SAR     AR1,TIMELO    ; 1
           LRLK    AR0,61440     ; 2
           CMPR    3             ; 1
           BBNZ    INCDON        ; 3      ; If at max, reset seed and time
           LARK    AR1,1         ; 1      ; This allows a periodic random
           SAR     AR1,SEED      ; 1      ;  sequence with period 61440 hops
           LARK    AR1,0         ; 1
           SAR     AR1,SEED+1    ; 1
           SAR     AR1,TIMELO    ; 1
           SAR     AR1,TIMEHI    ; 1 [16]
```

117

```
INCDON:
          RET                 ; 3ret+3cal[19] 19 cycles @ 100 ns = 1.9 us



;=============================================================================
;
;          STPTIM - Advance time 16 frames for next downlink burst
;
;-----------------------------------------------------------------------------
;
;                   in:   -
;                   out:  TIMELO
;                   used: ACC
;
;                   cycles:
;                         10              (includes call/return)
;                   time:
;                         1.0 us          (includes call/return)
;
;
;-----------------------------------------------------------------------------
;
STPTIM:
          ZALS      TIMELO      ; 1 (4)          ; Note that time has already been
          ADLK      5120-6      ; 2              ;  advanced by 6 hops.
          SACL      TIMELO      ; 1 [4]

          RET                 ; 3ret+3cal[10] 10 cycles @ 100 ns = 1.0 us



;=============================================================================
;
;          LDHOPN - load hop number from the random number
;
;-----------------------------------------------------------------------------
;
;                   in:   SEED
;                   out:  HOPNUM
;                   used: ACC
;
;                   cycles:
;                         11              (includes call/return)
;                   time:
;                         1.1 us          (includes call/return)
;
;-----------------------------------------------------------------------------
;
LDHOPN:
          ZALS      SEED        ; 1 (5)
          SACL      HOPNUM      ; 1
          ZALS      SEED+1      ; 1
          ANDK      00FFH       ; 1     ; Mask off upper 8 bits giving 24 bits
          SACL      HOPNUM+1    ; 1 [5]

          RET                 ; 3ret+3cal[11] 11 cycles @ 100 ns = 1.1 us



;=============================================================================
;
;          SETDEF - sets default starting values for calculation parameters
;
;-----------------------------------------------------------------------------
;
;                   in:   <nil>
;                   out:  HOPNUM,BWSCAL,FSKCHN,FCSPAC,FBASE,FOFFST,DOPFCT,FLOCOM
;                         TIMELO,TIMEHI,FLOSCI,FLAGS
;                   used: -
```

```
;
;               cycles:
;                       116             (includes call/return)
;               time:
;                       11.6 us         (includes call/return)
;
;-------------------------------------------------------------------------
;

                .data
                        ; GT Uplink
GUHOPN:         .long   0               ; Hop number 24 bits 0-16777215
                                        ;   max value 16777215
GUBWSC:         .long   250000          ; Hopping BW scale
                                        ;   gives scaled BW of 0-500000 [100Hz]
GUFSKC:         .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
                                        ;   max composite value is 255
GUFCSP:         .long   360             ; Tone spacing 360 [100Hz] = 36kHz
                                        ;   max value is 3.2768MHz
GUFBAS:         .long   445950000       ; Base freq of hop range [100Hz] = 44.595GHz
                                        ;   max value is 99.9GHz
GUFOFF:         .long   0               ; Guard offset from base [100Hz] = 0kHz
                                        ;   max value is 99.9GHz
GUDOPF:         .long   0               ; Doppler factor (x2^32)
                                        ;   max value is +42950 = +1.00000761449E-05
                                        ;     gives 400kHz on 40GHz
                                        ;   smallest delta is 1 =  2.32830643654E-10
                                        ;     gives 9Hz on 40GHz
                                        ;     (note best resolution is 100Hz)
                                        ;   min value is -42950 = -1.00000761449E-05
GUFLOC:         .long   439500000       ; Upconverter LO [100Hz] = 43.95GHz for Comstron
                                        ;   this value will not be sent to synth
                                        ;   so it is subtracted off before BCD
GUFLOS:         .long   431100000       ; Upconverter LO [100Hz] = 43.11GHz for Sciteq
                                        ;   this value will not be sent to synth
                                        ;   so it is subtracted off before conversion
GUFLAG:         .long   VNODIV+VSTOP    ; Flags (no Sciteq divide by two, startup
                                        ;   in STOP mode)

                        ; GT Downlink
GDHOPN:         .long   0               ; Hop number
GDBWSC:         .long   250000          ; Hopping BW scale 0-500000 [100Hz]
GDFSKC:         .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
GDFCSP:         .long   360             ; Tone spacing 36kHz
GDFBAS:         .long   445950000       ; Base freq of hopping range [100Hz] = 44.595GHz
GDFOFF:         .long   0               ; Guard offset from base [100Hz] = 0kHz
GDDOPF:         .long   0               ; Doppler factor (x2^32)
GDFLOC:         .long   439500000       ; Upconverter LO [100Hz] = 43.95GHz
GDFLOS:         .long   431100000       ; Upconverter LO [100Hz] = 43.11GHz for Sciteq
GDFLAG:         .long   VNODIV+VSTOP    ; Flags (no Sciteq divide by 2, STOP mode)

                        ; Payload Uplink
PUHOPN:         .long   0               ; Hop number 24 bits 0-16777215
PUBWSC:         .long   250000          ; Hopping BW Scale 0-500000 [100Hz]
PUFSKC:         .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
PUFCSP:         .long   360             ; Tone spacing 36kHz
PUFBAS:         .long   076200000       ; Base freq of hopping range [100Hz] = 7.62GHz
PUFOFF:         .long   0               ; Guard offset from base [100Hz] = 0kHz
PUDOPF:         .long   0               ; Doppler factor (x2^32)
PUFLOC:         .long   040950000       ; Upconverter LO [100Hz] = 4.095GHz for Comstron
PUFLOS:         .long   040950000       ; Upconverter LO [100Hz] = 4.095GHz for Sciteq
PUFLAG:         .long   VDIVID+VSTOP    ; Flags (Sciteq divide by 2, STOP mode)

                        ; Payload Downlink
PDHOPN:         .long   0               ; Hop number
PDBWSC:         .long   250000          ; Hopping BW scale 0-500000 [100Hz]
PDFSKC:         .long   0               ; FSK(3LSB 0-7)/Channel(5MSB 0-31)
PDFCSP:         .long   360             ; Tone spacing 36kHz
PDFBAS:         .long   076200000       ; Base freq of hopping range [100Hz] = 7.62GHz
```

119

```
PDFOFF:   .long    0                ; Guard offset from base [100Hz] = 0kHz
PDDOPF:   .long    0                ; Doppler factor (x2^32)
PDFLOC:   .long    040950000        ; Upconverter LO [100Hz] = 4.095GHz for Comstron
PDFLOS:   .long    040950000        ; Upconverter LO [100Hz] = 4.095GHz for Sciteq
PDFLAG:   .long    VDIVID+VSTOP     ; Flags (Sciteq divide by 2, STOP mode)

ALLTLO:   .long    0                ; Time (low longword) in number of hops since
ALLTHI:   .long    0                ;        (high longword) time 0.


          .text
SETDEF:
          BIT      STAT,BGT      ; 1 (11)  ; Check pyaload vs ground terminal
          BBZ      PAYLOAD       ; 3
          BIT      STAT,BUP      ; 1        ; Check U/L vs D/L for ground terminal
          BBZ      GTDOWN        ; 3
          B        GTUP          ; 3 [max 11]
PAYLOAD:
          BIT      STAT,BUP      ; 1        ; Check U/L vs D/L for payload
          BBZ      PAYDOW        ; 3
          B        PAYUP         ; 3 [max 11]

GTUP:
          BLKP     GUHOPN,HOPNUM ; (83)  ; Load GT Uplink table into RAM
          BLKP     GUHOPN+1,HOPNUM+1
          BLKP     GUBWSC,BWSCAL
          BLKP     GUBWSC+1,BWSCAL+1
          BLKP     GUFSKC,FSKCHN
          BLKP     GUFSKC+1,FSKCHN+1
          BLKP     GUFCSP,FCSPAC
          BLKP     GUFCSP+1,FCSPAC+1
          BLKP     GUFBAS,FBASE
          BLKP     GUFBAS+1,FBASE+1
          BLKP     GUFOFF,FOFFST
          BLKP     GUFOFF+1,FOFFST+1
          BLKP     GUDOPF,DOPFCT
          BLKP     GUDOPF+1,DOPFCT+1
          BLKP     GUFLOC,FLOCOM
          BLKP     GUFLOC+1,FLOCOM+1
          BLKP     GUFLOS,FLOSCI
          BLKP     GUFLOS+1,FLOSCI+1
          BLKP     GUFLAG,FLAGS
          BLKP     GUFLAG+1,FLAGS+1
          B        COMMON              ; [94]

GTDOWN:
          BLKP     GDHOPN,HOPNUM ; (83)  ; Load GT Downlink table into RAM
          BLKP     GDHOPN+1,HOPNUM+1
          BLKP     GDBWSC,BWSCAL
          BLKP     GDBWSC+1,BWSCAL+1
          BLKP     GDFSKC,FSKCHN
          BLKP     GDFSKC+1,FSKCHN+1
          BLKP     GDFCSP,FCSPAC
          BLKP     GDFCSP+1,FCSPAC+1
          BLKP     GDFBAS,FBASE
          BLKP     GDFBAS+1,FBASE+1
          BLKP     GDFOFF,FOFFST
          BLKP     GDFOFF+1,FOFFST+1
          BLKP     GDDOPF,DOPFCT
          BLKP     GDDOPF+1,DOPFCT+1
          BLKP     GDFLOC,FLOCOM
          BLKP     GDFLOC+1,FLOCOM+1
          BLKP     GDFLOS,FLOSCI
          BLKP     GDFLOS+1,FLOSCI+1
          BLKP     GDFLAG,FLAGS
          BLKP     GDFLAG+1,FLAGS+1
          B        COMMON              ; [94]
```

```
PAYUP:
        BLKP    PUHOPN,HOPNUM  ; (83)   ; Load Payload Uplink table into RAM
        BLKP    PUHOPN+1,HOPNUM+1
        BLKP    PUBWSC,BWSCAL
        BLKP    PUBWSC+1,BWSCAL+1
        BLKP    PUFSKC,FSKCHN
        BLKP    PUFSKC+1,FSKCHN+1
        BLKP    PUFCSP,FCSPAC
        BLKP    PUFCSP+1,FCSPAC+1
        BLKP    PUFBAS,FBASE
        BLKP    PUFBAS+1,FBASE+1
        BLKP    PUFOFF,FOFFST
        BLKP    PUFOFF+1,FOFFST+1
        BLKP    PUDOPF,DOPFCT
        BLKP    PUDOPF+1,DOPFCT+1
        BLKP    PUFLOC,FLOCOM
        BLKP    PUFLOC+1,FLOCOM+1
        BLKP    PUFLOS,FLOSCI
        BLKP    PUFLOS+1,FLOSCI+1
        BLKP    PUFLAG,FLAGS
        BLKP    PUFLAG+1,FLAGS+1
        B       COMMON              ; [94]

PAYDOW:
        BLKP    PDHOPN,HOPNUM  ; (83)   ; Load Payload Downlink table into RAM
        BLKP    PDHOPN+1,HOPNUM+1
        BLKP    PDBWSC,BWSCAL
        BLKP    PDBWSC+1,BWSCAL+1
        BLKP    PDFSKC,FSKCHN
        BLKP    PDFSKC+1,FSKCHN+1
        BLKP    PDFCSP,FCSPAC
        BLKP    PDFCSP+1,FCSPAC+1
        BLKP    PDFBAS,FBASE
        BLKP    PDFBAS+1,FBASE+1
        BLKP    PDFOFF,FOFFST
        BLKP    PDFOFF+1,FOFFST+1
        BLKP    PDDOPF,DOPFCT
        BLKP    PDDOPF+1,DOPFCT+1
        BLKP    PDFLOC,FLOCOM
        BLKP    PDFLOC+1,FLOCOM+1
        BLKP    PDFLOS,FLOSCI
        BLKP    PDFLOS+1,FLOSCI+1
        BLKP    PDFLAG,FLAGS
        BLKP    PDFLAG+1,FLAGS+1
        B       COMMON              ; [94]

COMMON:
        BLKP    ALLTLO,TIMELO       ; (16)
        BLKP    ALLTLO+1,TIMELO+1
        BLKP    ALLTHI,TIMEHI
        BLKP    ALLTHI+1,TIMEHI+1   ; [110]

        RET         ; 3ret+3cal[116] 116 cycles @ 100 ns = 11.6 us


;==============================================================================
;
;       PRETTY - this cycles through the digits on the Comstron display at
;               powerup.  From 1111111100.0 to 9999999900.0 and then leaves
;               it at midband.
;
;------------------------------------------------------------------------------
;
;               in:   <nil>
;               out:  <nil>
;               used: AR1,ANSWER
;
;               cycles:
```

121

```
;                    47160442         (includes call/return)
;                time:
;                    4.7 s           (includes call/return)
;
;--------------------------------------------------------------------

PRETTY:
        LARP    AR1             ; 1 (14)
        LARK    AR1,9           ; 1
        LALK    VERLO           ; 2
        SACL    ANSWER          ; 1
        LALK    VERHI           ; 2
        SACL    ANSWER+1        ; 1
        OUT     ANSWER,SYNTHL   ; 2
        OUT     ANSWER+1,SYNTHH         ; 2
        ZAC                     ; 1
        SACL    ANSWER          ; 1 [14]
PLOOP:
        LALK    20000           ; } 2 Outer loop each = 5240017,total = 47160027
        SACL    ANSWER+1        ; } 1
POSLP:                          ; }     } Middle loop each = 262, total = 5240000
        RPTK    255             ; }     }   } 1 Inner loop total = 257
        ZALS    ANSWER+1        ; }     }   } 1
        SUBK    1               ; }     } 1
        SACL    ANSWER+1        ; }     } 1
        BNZ     POSLP           ; }     } 3

        LALK    CPRETY          ; } 2
        ADD     ANSWER          ; } 1
        SACL    ANSWER          ; } 1
        OUT     ANSWER,SYNTHL   ; } 2
        OUT     ANSWER,SYNTHH   ; } 2
        BANZ    PLOOP,*-        ; } 3    ;Test, then decrement AR1
                                ; [47160041]

        LACK    0               ; 1 (395)  ;Set hop number to mid-band
        SACL    HOPNUM          ; 1
        LALK    080H            ; 2
        SACL    HOPNUM+1        ; 1
        CALL    CALC            ; 386
        OUT     ANSWER,SYNTHL   ; 2
        OUT     ANSWER+1,SYNTHH         ; 2 [47160436]

        RET            ; 3ret+3cal[47160442] 47160442 cycles @ 100 ns = 4.7 s

        .end
```

3.3   Calculation Subroutines (HSC_CALC.ASM)

```
;==============================================================================
;          HSC_CALC.ASM - Hopping Synthesizer Controller calculation routines
;------------------------------------------------------------------------------
; Version V960501.03  01 May 96
;------------------------------------------------------------------------------
; No code change but comments modified and key labels were renamed 21 Jun 96.
;------------------------------------------------------------------------------
          .length  55
          .width   120
          .global  CALC,SCICVT,MUL32,BCD32,BCDINI
          .global  ANSWER,SEED,RAN_A,ZERO,ONE,COFFFF,CO1000,CULBAS
          .global  HOPNUM,BWSCAL,FSKCHN,FCSPAC,FBASE,FOFFST,DOPFCT,FLOCOM
          .global  TIMELO,TIMEHI,FLOSCI,FLAGS
          .global  X0,X1,Y0,Y1,Z0,Z1,Z2,Z3,SIGN,TMP
          .global  CMD,DATA1,DATA2,WPNT,STAT,MODE
          .global  HTAB,HEND,LTAB,LEND

; Status register bits
BGT:      .set     7          ; GT/NPayload mode              (7 -> bit 8)
BUP:      .set     6          ; Uplink/NDownlink              (6 -> bit 9)
BINTCK:   .set     5          ; Internal/NExternal hop clock  (5 -> bit 10)
BCOMST:   .set     4          ; Comstron/NSciteq              (4 -> bit 11)
B1GHZ:    .set     3          ; RF off (1 GHz offset)/RF on (no offset)
BBIO:     .set     2          ; Command ready                 (2 -> bit 13)
BSPARE:   .set     1          ; -                             (1 -> bit 14)
BHOP:     .set     0          ; Hop clock                     (0 -> bit 15)

; Flags
FSCID2:   .set     15         ; Divide Sciteq frequency by two (15 -> bit 0)
FRUN:     .set     14         ; Startup in RUN mode           (14 -> bit 1)


;==============================================================================
;
;          CALC - calculates synth output value from hop number
;
;------------------------------------------------------------------------------
;
;                  in:
;                      HOPNUM+1,HOPNUM - 24 bit hop number
;                      FSKCHN - FSK tone (3 lsb) and channel number (5 msb)
;                      BWSCAL,FCSPAC,FBASE,FOFFST,DOPFCT,FLOCOM,FLOSCI,FLAGS
;                              - as describes in MAIN
;                  out:
;                      ANSWER and ACC - 32 bit result
;                  used:
;                      ARP,AR0,AR1,ACC,T,P
;
;                  cycles:
;                      386             (includes call/return)
;                  time:
;                      38.6us          (includes call/return)
;
;------------------------------------------------------------------------------

          .text
CALC:
          ZALS     HOPNUM      ; 1 (69)  ; Scale hop number to hopping BW
          SACL     X0          ; 1
          ZALS     HOPNUM+1    ; 1
          SACL     X1          ; 1
          ZALS     BWSCAL      ; 1
          SACL     Y0          ; 1
          ZALS     BWSCAL+1    ; 1
          SACL     Y1          ; 1
          CALL     MUL32       ; 54
```

123

```
        LAC     Z1,9        ; 1        ; Shift to result right by 2^23
        SACH    X0          ; 1
        LAC     Z3,9        ; 1
        SACL    X1          ; 1
        ZALH    X1          ; 1
        ADD     Z2,9        ; 1
        ADD     X0          ; 1 [69]

        LT      FSKCHN      ; 1 (3) ; Calculate FSK and channel freq
        MPY     FCSPAC      ; 1        ;  and add to hop frequency
        APAC                ; 1 [72]

        ADDS    FBASE       ; 1 (2) ; Add in base freq
        ADDH    FBASE+1     ; 1 [74]

        ADDS    FOFFST      ; 1 (2) ; Add in offset from base
        ADDH    FOFFST+1    ; 1 [76]

        SACL    X0          ; 1 (64) ; Compute doppler and add in
        SACH    X1          ; 1
        ZALS    DOPFCT      ; 1
        SACL    Y0          ; 1
        ZALS    DOPFCT+1    ; 1
        SACL    Y1          ; 1
        CALL    MUL32       ; 54
        ZALH    X1          ; 1
        ADDS    X0          ; 1
        ADDS    Z2          ; 1
        ADDH    Z3          ; 1 [140]

        BIT     STAT,BCOMST ; 1 (4) ; Check if Comstron or Sciteq
        BBZ     SCITEQ      ; 3 [144]

COMSTR:                     ; (213) ; Format for Comstron
        SUBS    FLOCOM      ; 1        ; Subtract off the uplink LO freq
        SUBH    FLOCOM+1    ; 1
        SACL    X0          ; 1        ; Convert to BCD
        SACH    X1          ; 1
        CALL    BCD32       ; 198
        ZALH    Z1          ; 1
        ADDS    Z0          ; 1
        BIT     STAT,B1GHZ  ; 1        ; If 1GHz offset, XOR 1GHz bit
        BBZ     FINISH      ; 3
        XORK    2000H,15    ; 2
        B       FINISH      ; 3 [357/144]

SCITEQ:                     ; (234) ; Format for Sciteq
        SUBS    FLOSCI      ; 1        ; Subtract off the uplink LO freq
        SUBH    FLOSCI+1    ; 1
        BIT     FLAGS,FSCID2 ; 1       ; Divide frequency by 2 if set
        BBZ     SCIL1       ; 3
        SFR                 ; 1
SCIL1:  SACL    X0          ; 1
        SACH    X1          ; 1
        CALL    SCICVT      ; 215
        ZALH    Z1          ; 1
        ADDS    Z0          ; 1
        BIT     STAT,B1GHZ  ; 1        ; If 1GHz offset=RF off, set 1GHz bit
        BBZ     FINISH      ; 3
        ADDH    C01000      ; 1
        B       FINISH      ; 3 [357/378]

FINISH:
        SACL    ANSWER      ; 1 (2)
        SACH    ANSWER+1    ; 1 [359/380]

        RET             ; 3ret+3call[365/386] 386 cycles @ 100ns = 38.6 us
```

124

```
;===========================================================================
;
;                         S C I C V T
;
;--------------------------------------------------------------------------
;
;                 in:
;                       X1,X0 - binary value, 0-9999999 (5F5E0FFH)
;                 out:
;                       Z1,Z0 - Sciteq value
;                 used:
;                       ACC,ANSWER
;
;                 cycles:
;                       215                 (includes call/return)
;                 time:
;                       21.5 us             (includes call/return)
;
;--------------------------------------------------------------------------
;

SCICVT:
        LALK    304,15          ; 2 (10)  ; Set up a constant 1 GHz = 10000000
        ADLK    38528 ; 2
        SACL    Z0      ; 1
        SACH    Z1      ; 1
        ZALH    X1      ; 1        ; Subtract 1 GHz from the desired frequency
        ADDS    X0      ; 1        ; to get the value needed to set the synth
        SUBS    Z0      ; 1
        SUBH    Z1      ; 1 [10]

        SACL    X0      ; 1 (4) ; Store frequency for division and remainder
        SACH    X1      ; 1        ;    calculations
        SACL    ANSWER          ; 1
        SACH    ANSWER+1  ; 1 [14]

        LALK    17180 ; 2 (60) ; Get the coarse bits by dividing by 25 MHz
        SACL    Y0      ; 1        ; Divide by 250000 is equivalent to multiply
        SACH    Y1      ; 1        ;    by 17180 and divide by 2^32
        CALL    MUL32 ; 54
        ZALS    Z2      ; 1
        SACL    TMP     ; 1 [74]

        SACL    X0      ; 1 (64)  ; Get remainder of the division for the fine
        SACH    X1      ; 1        ;    bits
        LALK    15625,4         ; 2        ; Value is 250000
        SACL    Y0      ; 1
        SACH    Y1      ; 1
        CALL    MUL32 ; 54
        ZALH    ANSWER+1  ; 1
        ADDS    ANSWER          ; 1
        SUBS    Z0      ; 1
        SUBH    Z1      ; 1 [138]

        ADLK    50000 ; 2 (2) ; Add in 5 MHz to make effective 1 GHz offset

        SACL    X0      ; 1 (66) ; Change LSB from 100 Hz to 33.3786010742 Hz
        SACH    X1      ; 1        ; Multiply by 2.99593142857*2^16 = 196341
        LALK    4,15 ; 2        ; Value is 196341
        ADLK    65269 ; 2
        SACL    Y0      ; 1
        SACH    Y1      ; 1
        CALL    MUL32 ; 54
        ZALH    Z2      ; 1
        ADDS    Z1      ; 1
        SACL    Z0      ; 1
        SACH    Z1      ; 1 [206]

        LAC     TMP,4 ; 1 (3) ; Place coarse bits above fine bits
```

125

```
            ADD       Z1      ; 1
            SACL      Z1      ; 1 [209]

            RET               ; 3ret+3call[215] 215 cycles @ 100ns = 21.5 us

;============================================================================
;
;           BCD32 - binary to BCD conversion 32 bit output (8 digits)
;
;----------------------------------------------------------------------------
;
;                 in:
;                     X1,X0 - binary value, 0-9999999 (5F5E0FFH)
;                 out:
;                     Z1,Z0 - BCD value
;                 used:
;                     ARP,AR0,AR1,ACC
;
;                 cycles:
;                       198              (includes call/return)
;                 time:
;                       19.8us           (includes call/return)
;
;----------------------------------------------------------------------------
;
            .text
BCD32:
            LARP      1          ; 1      ; Point to BCD table
            LRLK      AR1,HTAB   ; 2
            LRLK      AR0,HEND   ; 2
            ZALH      X1         ; 1      ; Load in value to be converted
            ADDS      X0         ; 1 [7]

LOOP:
            SUBS      *+         ; 1(9)   ; Determine if 8 position is required
            SUBH      *-         ; 1      ;  by subtracting and checking borrow.
            BC        DID8       ; 3      ;  All operations 32 bits
            ADDS      *+         ; 1      ; Not required, so add constant back
            ADDH      *-         ; 1
            RC                   ; 1      ; Clear carry for 0 in 8 position
DID8:       ROL                  ; 1      ; Double ACC and add carry

            SUBS      *+         ; 1(9)   ; Determine if 4 position is required
            SUBH      *-         ; 1
            BC        DID4       ; 3
            ADDS      *+         ; 1
            ADDH      *-         ; 1
            RC                   ; 1
DID4:       ROL                  ; 1

            SUBS      *+         ; 1(9)   ; Determine if 2 position is required
            SUBH      *-         ; 1
            BC        DID2       ; 3
            ADDS      *+         ; 1
            ADDH      *-         ; 1
            RC                   ; 1
DID2:       ROL                  ; 1

            SUBS      *+         ; 1(9)   ; Determine if 1 position is required
            SUBH      *-         ; 1
            BC        DID1       ; 3
            ADDS      *+         ; 1
            ADDH      *-         ; 1
            RC                   ; 1
DID1:       ROL                  ; 1

            ADRK      2          ; 1(5) 1 loop = 41, 4 loops = 164
            CMPR      0          ; 1
```

126

```
        BBZ      LOOP          ; 3 [171]        ; Do for high 16 bits

        SACL     Z1            ; 1(5)           ; Do the low 16 bit conversion
        SACH     Z0            ; 1
        LAC      Z0,15         ; 1
        LRLK     AR1,LTAB      ; 2 [176]

        SUBC     *             ; 1(16) ; Conditional subtraction, similar to
        SUBC     *             ; 1     ;  above, but only 16 bits.
        SUBC     *             ; 1
        SUBC     *+            ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *+            ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *+            ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *             ; 1
        SUBC     *+            ; 1
        SACL     Z0            ; 1 [192]        ; Store lower 16 bits

        RET                    ; 3ret+3call[198] 198 cycles @ 100ns = 19.8 us


;==============================================================================
;
;        BCDINI - initialization for binary to BCD conversion (BCD32)
;
;------------------------------------------------------------------------------
;
;                in:    <nil>
;                out:
;                        HTAB - table of long constants (8 words)
;                        LTAB - table of short constants (4 words)
;                used:
;                        ARP,AR1
;
;                cycles:
;                        25              (includes call/return)
;                time:
;                        2.5us           (includes call/return)
;
;------------------------------------------------------------------------------

        .data

HTABLE: .long    80000000              ; Constant table for 32 bit operations
        .long    8000000*16            ;  to compute high 16 bits.
        .long    800000*16*16
        .long    80000*16*16*16

LTABLE: .word    8000                  ; Constant table for 16 bit operations
        .word    800*16                          ; to compute low 16 bits
        .word    80*16*16
        .word    8*16*16*16

        .text
BCDINI:
        LARP     1             ; 1      ; Load high table into RAM
        LRLK     AR1,HTAB      ; 2
        RPTK     7             ; 1
        BLKP     HTABLE,*+     ; 11
        LRLK     AR1,LTAB      ; 2      ; Load low table into RAM
        RPTK     3             ; 1
```

127

```
        BLKP    LTABLE,*+      ; 7 [25]

        RET                    ; 3ret+3call[25] 25 cycles @ 100ns = 2.5 us


;==============================================================================
;
;         MUL32 - 32 x 32 = 64 bit signed multiply
;
;------------------------------------------------------------------------------
;
;                  in:
;                       X1,X0 - value1
;                       Y1,Y0 - value2
;                  out:
;                       Z3,Z2,Z1,Z0 - result
;                  used:
;                       ACC,T,P
;
;                  cycles:
;                       54                  (includes call/return)
;                  time:
;                       5.4us               (includes call/return)
;
;------------------------------------------------------------------------------

        .text
MUL32:
        ZALS    X1     ; 1(3)  ; Determine sign of product
        XOR     Y1     ; 1
        SACH    SIGN,1        ; 1[3]

ABSX:   ZALH    X1     ; 1(5)  ; Take absolute value of X
        ADDS    X0     ; 1
        ABS            ; 1
        SACH    X1     ; 1
        SACL    X0     ; 1[8]

ABSY:   ZALH    Y1     ; 1(5)  ; Take absolute value of Y
        ADDS    Y0     ; 1
        ABS            ; 1
        SACH    Y1     ; 1
        SACL    Y0     ; 1[13]

MULT:   LT      X0     ; 1(4)  ; X0*Y0 -> Z1,Z0
        MPYU    Y0     ; 1
        SPL     Z0     ; 1
        SPH     Z1     ; 1[17]

        MPYU    Y1     ; 1(2)  ; X0*Y1 -> acc
        LTP     X1     ; 1[19]

        MPYU    Y0     ; 1(2)  ; X0*Y1 + (X0*Y0)H -> acc
        ADDS    Z1     ; 1[21]

        MPYA    Y1     ; 1(11) ; X1*Y0 + X0*Y1 + (X0*Y0) -> Z2,Z1
        SACL    Z1     ; 1
        SACH    Z2     ; 1
        ZALS    Z2     ; 1        ; X1*Y1 + Z2 + carry -> Z3,Z2
        BNC     SUM    ; 3
        ADDH    ONE    ; 1
SUM:    APAC           ; 1
        SACL    Z2     ; 1
        SACH    Z3     ; 1[32]

        LAC     SIGN   ; 1(4)  ; Fix sign
        BZ      DONE   ; 3[36]
```

128

```
        ZALH    Z1      ; 1(12)  ;Negate product
        ADDS    Z0      ; 1
        CMPL            ; 1
        ADD     ONE     ; 1
        SACL    Z0      ; 1
        SACH    Z1      ; 1
        ZALS    Z2      ; 1
        ADDH    Z3      ; 1
        CMPL            ; 1
        ADDC    ONE     ; 1
        SACL    Z2      ; 1
        SACH    Z3      ; 1[48]

DONE:   RET             ; 3ret+3call[54] 54 cycles @ 100ns = 5.4 us

        .end
```

3.4 Random Number Generation Routines (HSC_RAN.ASM)

```
;==============================================================================
;        HSC_RAN.ASM - Hopping Synthesizer Controller random number routines
;------------------------------------------------------------------------------
; Version V960501.03  01 May 96
;------------------------------------------------------------------------------
; No code change but comments modified and key labels were renamed 21 Jun 96.
;------------------------------------------------------------------------------
           .length  55
           .width   120
           .global  RAN,RANINI,RANJMP
           .global  ANSWER,SEED,RAN_A,ZERO,ONE,COFFFF,CO8000,CULBAS
           .global  HOPNUM,BWSCAL,FSKCHN,FCSPAC,FBASE,FOFFST,DOPFCT,FLOCOM
           .global  TIMELO,TIMEHI,FLOSCI,FLAGS
           .global  X0,X1,Y0,Y1,Z0,Z1,Z2,Z3,SIGN,TMP


;==============================================================================
;
;        RAN - computes next random number
;
;------------------------------------------------------------------------------
;
;                in:   RAN_A
;                out:
;                      SEED+1,SEED and ACC
;                used:
;                      ACC,P,T
;
;                cycles:
;                      24                (includes call/return)
;                time:
;                      2.4 us            (includes call/return)
;------------------------------------------------------------------------------

           .text

RAN:
           LT       RAN_A  ; 1 (4)  ; Multiply low seed by constant
           MPYU     SEED   ; 1
           SPL      Z0     ; 1
           SPH      Z1     ; 1 [4]

           ZALS     Z1     ; 1 (12) ; Load in partial sum /2^16
           MPYU     SEED+1       ; 1      ; Multiply high seed by constant
           APAC            ; 1      ; Add in
           ROL             ; 1      ; Shift one bit left and mask off all but
           SACH     Z2     ; 1      ;  lower 31 bits then shift back
           AND      COFFFF       ; 1
           ROR             ; 1
           SACL     Z3     ; 1
           ZALH     Z3     ; 1      ; Load in partial sum
           ADDS     Z0     ; 1      ; Add in from original low multiply
           ADDS     Z2     ; 1      ; Add in high word (see modified method in
           ADDC     ZERO   ; 1 [16] ;  the paper) and add in carry (overflow)

           SACL     SEED   ; 1 (2)  ; Save new seed
           SACH     SEED+1       ; 1 [18]

           RET             ; 3ret+3call[24] 24 cycles @ 100ns = 2.4 us


;==============================================================================
;
```

```
;        RANINI - intializes values for pseudo random generation
;
;------------------------------------------------------------------------
;
;              in:    <nil>
;              out:
;                     ZERO,ONE,COFFFF,SEED,RAN_A
;              used:
;                     ACC
;
;              cycles:
;                     19              (includes call/return)
;              time:
;                     1.9 us          (includes call/return)
;
;------------------------------------------------------------------------

RANINI:
        LACK     0      ; 1 (7) ; Initialize constants
        SACL     ZERO   ; 1
        LACK     1      ; 1
        SACL     ONE    ; 1
        LALK     0FFFFH        ; 2
        SACL     COFFFF        ; 1 [7]

        LALK     16807 ; 2 (3) ; A = 16807
        SACL     RAN_A ; 1 [10]

        LACK     1      ; 1 (3) ; Seed <- 00000001
        SACL     SEED   ; 1
        SACH     SEED+1        ; 1 [13]

        RET            ; 3ret+3call[19] 19 cycles @ 100ns = 1.9 us


;========================================================================
;
;        RANJMP - jump to certain place in random sequence
;
;------------------------------------------------------------------------
;
;              in:
;                     TIMELO - number to jump to
;              out:
;                     SEED+1,SEED and ACC
;              used:
;                     ARP,AR0,AR2,ACC,T,P
;
;              cycles:
;                     for 16 bits or less
;                     21+29/jmp               (includes call/return)
;              time:
;                     for 16 bits or less
;                     2.1us + 2.9us/jmp       (includes call/return)
;
;                     for 10000 jumps, time is 29ms
;
;------------------------------------------------------------------------

RANJMP:
        LARP     AR1           ; 1 (3) ; Initialize seed to 1
        LARK     AR1,1         ; 1
        SAR      AR1,SEED      ; 1
        LARK     AR0,0         ; 1
        SAR      AR0,SEED+1    ; 1 [5]

        LAR      AR1,TIMELO    ; 1 (5) ; Is lower start time 0 ?
        CMPR     0             ; 1
```

131

```
            BBNZ      CHKHI          ; 3 [8]
LO:
            CALL      RAN            ; 24 (29)  ; The lower 16 bits of the count
            SBRK      1              ; 1
            CMPR      0              ; 1
            BBZ       LO             ; 3 [8+29/]
CHKHI:
            LARK      AR0,0          ; 1 (7) ; Is upper start time 0 ?
            LAR       AR1,TIMELO+1   ; 1
            CMPR      0              ; 1
            BBNZ      DONE           ; 3 [15+29/]

            LARK      AR2,0          ; 1
PREPHI:
            LARP      AR2            ;1
HI:
            CALL      RAN            ; 24      ; One full loop for each of the
            SBRK      1              ; 1       ;   upper 16 bits of the count
            CMPR      0              ; 1
            BBZ       HI             ; 3

            LARP      AR1            ; 1       ; Check to see if all upper loops done
            SBRK      1              ; 1
            CMPR      0              ; 1
            BBZ       PREPHI                   ; 3
DONE:                                ; [15+29/    exclude high 16 bit loops]
            RET               ; 3ret+3call [21+29/] 21 cycles @ 100ns = 2.1 us

            .end
```

Appendix F: PC Test Program for HSC

1.   General

For the development and test of the Hopping Synthesizer Controller, the functions of the Synchronization Controller had to be emulated. A program was developed in 'C' to meet this requirement. The program, TESTHSC, runs on a PC and uses a digital I/O board installed on the ISA (Industry Standard Architecture) bus. The program allows any of the commands to be issued to the HSC as well as the ability to single-step the hop clock to allow for observation of the hopping pattern.

2.   Usage

When the program is started by typing TESTHSC, the version number is displayed followed by a prompt. The HELP command can be used to get a list of valid commands and parameters. The VAL command gives the same list of commands but rather than a description, gives the hexadecimal equivalents - this is useful to find addresses for the READ command.

The prompt includes the display of the status of the NReady and Sync lines. These lines are checked once before the prompt is issued. Any change in the status will not be reflected until the next prompt is displayed. At the prompt, any of the commands can be entered in the form of a one-word command with an optional hexadecimal parameter. The valid commands are given in Section 4 of this appendix.

Usually during debugging, single-step hop clocking is required. To achieve this, the HSC must be configured for external clock. Each time the HOP command is given, the Hclk line will be pulsed. Note that while in RUN mode, the HSC will only process one command per hop, so commands must be interspersed with HOP commands or the HSC will not be ready. If in doubt, use the HOP command until the prompt reflects a ready status.

Another feature added for debugging is the sweep command. This command causes the synthesizer to sweep from the start frequency to the stop frequency in steps at the time interval specified. This command has two forms, SWEEP and SWEEP 0. After a SWEEP 0, the user is prompted for the sweep parameters and then the sweep occurs. SWEEP alone repeats the last sweep.

During debugging, it is important that one not exceed the frequency range of the synthesizer. The range of the Comstron is 10 MHz to 4 GHz (actually 3.9999999 GHz) whereas the Sciteq range is 1 GHz to 2 GHz (actually 1.99999999 GHz).

To end the program, type QUIT.

3.   Parallel Board

The parallel board used is the DT2817 made by Data Translation. This board

133

is an ISA bus digital I/O board supporting four 8-bit ports for a total of 32 digital lines. The 50 pin connector on the back is then connected through a custom cable detailed below to the Synchronization Controller interface (P2) of the HSC.

Table F1. DT2817 to Synchronization Controller interface cable.

DT2817 Pin	DT2817 Name	P2 Pin	P2 Name
1	Gnd	1	Gnd
2	Gnd	2	Gnd
3	P0D0	3	D0
4	P1D0	4	D8
5	P0D1	5	D1
6	P1D1	6	D9
7	P0D2	7	D2
8	P1D2	8	D10
9	P0D3	9	D3
10	P1D3	10	D11
11	P0D4	11	D4
12	P1D4	12	D12
13	P0D5	13	D5
14	P1D5	14	D13
15	P0D6	15	D6
16	P1D6	16	D14
17	P0D7	17	D7
18	P1D7	18	D15
19-20	*unused*	-	-
21	Gnd	19	Gnd
22	Gnd	20	Gnd
23	Gnd	25	Gnd
24	Gnd	26	Gnd
25-32	*unused*	-	-
33	P2D0	21	Hclk
34	P3D0	23	Sync
35	P2D1	22	Strobe
36	P3D1	24	NReady
37-50	*unused*	-	-

The base address for the board is 228h. It occupies the area 228h to 22Ch in the PC I/O space. More details on programming the board can be found in [8]. The I/O port usage for TESTHSC is shown in Fig. F1.

Addr	Port	Use	Bit Fields					
229	0	Data Low Out			Data (lower 8 bits)			
22A	1	Data High Out			Data (upper 8 bits)			
22B	2	Handshake Out			unused		Strobe	HClk
22C	3	Handshake In			unused		Ready	Sync

Bit 7   Bit 0

Fig. F1. Parallel board port usage.

4. Commands

Most of the commands used in TESTHSC are the same as the ones detailed in Appendix B. All commands can be classified as unchanged, underscored, multiple parameter and test program commands. Commands are modified to force the format for parsing to consist of a single word command followed by an optional hex parameter.

4.1 Unchanged Commands

The following commands have the same format and parameters as described in the HSC Command Reference:

DEFAULTS
DLSYNC
FCALC
READ
RUN
STOP
ULSYNC

4.2 Underscored Commands

The following commands have the same parameters as detailed in the HSC Command Reference, but to simplify parsing in the program, the keywords are joined by an underscore to make a single keyword:

CHANGE_HOP
CHANGE_IMMEDIATE
LOAD_BASE
LOAD_BWSCALE
LOAD_DOPFACT
LOAD_FCSPACE
LOAD_FLAGS
LOAD_FSKCHAN
LOAD_HOP
LOAD_LATCH

135

```
LOAD_LOCOM
LOAD_LOSCI
LOAD_OFFSET
LOAD_SEED
LOAD_TIMELO
LOAD_TIMEHI
```

4.3   Multiple Parameter Commands

Where a command in the HSC Command Reference has more than one parameter, they are combined to form a composite parameter for TESTHSC. Below are the commands with composite parameters and the values to be used:

DLGO

    0-3 is for Hops 0-3 of Burst 0
    4 is for Hop 0 of Burst 1

ULGO

    0-F is for Hops 0-15 of Burst 0
    10-1F is for Hops 0-15 of Burst 1

4.4   Test Program Commands

There are some commands specific to TESTHSC and these are detailed below:

HELP

    Display a list of valid commands and ranges

HIGH

    This is a synonym for LOAD_LATCH FFFFFF

LOW

    This is a synonym for LOAD_LATCH 0

MID

    This is a synonym for LOAD_LATCH 800000

HOP or <cr>

    This command (or carriage return with no command) causes the hop clock line to be strobed. This is used when single-stepping the hop clock.

SWEEP or SWEEP 0

    SWEEP with a parameter of 0 causes the program to prompt the user for a new set of sweep parameters (start frequency, stop frequency, frequency step and time step). After the parameters are set, a sweep is then

136

performed. SWEEP with no parameters repeats the previous sweep.

VAL
Display a list of valid commands and their hexadecimal values.

QUIT or Q or EXIT or E
Terminates the TESTHSC program.

5. Listings

The software was compiled and linked on a Dell System 433E running under DOS 5.00 using Microsoft C 6.0 for DOS. The make file used to generate the test program follows:

```
testhsc.exe:      testhsc.obj
        link testhsc;

testhsc.obj:      testhsc.c
        cl /c testhsc.c
```

The software listing for TESTHSC follows.

```
//=========================================================================
//                TESTHSC.C - Hopping Synthesizer Controller PC Testing Program
//-------------------------------------------------------------------------
// Version V1.2  21 Aug 96
//-------------------------------------------------------------------------

#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

#define   VERSION   "V1.2 21 Aug 96"


//=========================================================================
//                       Bit Manipulation Macros
//-------------------------------------------------------------------------

#define SET(X,N) ((X) |= ((long) 1 << N))
#define RESET(X,N) ((X) &= ~((long) 1 << N))
#define TEST(X,N) ((X) & ((long) 1 << N))


//=========================================================================
//                       HSC Command Definitions
//-------------------------------------------------------------------------

#define MSTOP     0X0000                       /* Switch to STOP Mode           */
#define MDLSYN    0X4000                       /* Switch to DLSYNC Mode       */
#define  MULSYN   0X5000                       /* Switch to ULSYNC Mode       */
#define MRUN      0X8000                       /* Switch to RUN Mode            */

#define MDLGO     0X0060                       /* Select downlink sync hop frequency */
#define  MULGO    0X0200                       /* Select uplink sync probe frequency */
#define  MFCALC   0X5800                       /* Calculate one frames worth of    */
                                      /*  uplink sync probe frequencies       */

#define MLDLTC    0X0300                       /* Load latch directly           */
#define MLDSED    0X0302                       /* Load seed                     */
#define MLDHOP    0X0304                       /* Load hop number, calc and latch   */
#define MLDBWS    0X0306                       /* Load hopping bandwidth scale    */
#define MLDFSK    0X0308                       /* Load FSK/channel number        */
#define MLDFCS    0X030A                       /* Load FSK/channel spacing       */
#define MLDBAS    0X030C                       /* Load base frequency           */
#define MLDOFF    0X030E                       /* Load guard offset frequency     */
#define MLDDPF    0X0310                       /* Load Doppler factor            */
#define MLDLOC    0X0312                       /* Load Comstron LO value         */
#define MLDTLO    0X0314                       /* Load lower 32 bits of time     */
#define MLDTHI    0X0316                       /* Load upper 32 bits of time     */
#define  MLDLOS   0X0318                       /* Load Sciteq LO value           */
#define  MLDFLG   0X031A                       /* Load configuration control flags  */

#define MDEFLT    0X0400                       /* Select default values        */

#define MCGHOP    0X0500                       /* Change frequencies on hop      */
#define MCGIMM    0X0501                       /* Change frequencies immediately  */

#define MREAD     0X2000                       /* Read TMS320C25 RAM data        */


//=========================================================================
//                       DT2817 Parallel Port Definitions
//-------------------------------------------------------------------------

#define PARALL         0X0228       /* Base address of parallel card    */
#define CONTRL         PARALL       /* Control register                 */
#define LOBYTE         PARALL+1     /* Lower 8-bit of data              */
```

138

```
#define HIBYTE        PARALL+2      /* Upper 8-bit of data         */
#define HAND_OUT  PARALL+3     /* Handshaking output           */
    #define HCLK   0                  /* Hop clock line           */
    #define STROBE      1                      /* Strobe line          */
#define HAND_IN   PARALL+4    /* Handshaking input            */
    #define SYNC   0                  /* Sync line            */
    #define NREADY     1                      /* NReady line          */


//===============================================================================
//                      Global Variables
//-------------------------------------------------------------------------------

static int shadow;        // Shadow location for control register of DT2817


//===============================================================================
//                      Function Prototypes
//-------------------------------------------------------------------------------

void hop(void);                                     // Generate a pulse on hop clock
unsigned long intin(void);                          // Input a 16-bit value from HSC
void intout(unsigned long ival);        // Output a 16-bit value to HSC
void loadout(unsigned long cmd,unsigned long lval);        // Output a load command
void longout(unsigned long lval);                   // Output a 32-bit value to HSC
void parini(void);                                  // Initialize parallel port
int ready(void);                            // Wait for HSC ready
void strobe(void);                                  // Generate a pulse on strobe
int wait(int nticks);                               // Wait specified ticks (60 ms)


//===============================================================================
//                      Main Program
//-------------------------------------------------------------------------------

void main()
{
    char comd[80];                          // Current command string
    char str[80];                       // Input buffer string
    int ch;                                 // Character input
    int i;                                  // Integer index
    int nscan;                              // Number of fields scanned
    int ntick;                              // Number of ticks for sweep
    unsigned long lf;                       // Hop number for load in sweep
    unsigned long parm;                     // Current parameter value
    unsigned long stat;                     // Current handshaking in value
    unsigned long val;                      // Read value
    unsigned long val_hi;                   // Read value (upper 16 bits)
    unsigned long val_lo;                   // Read value (lower 16 bits)
    double f;
    double fbw;                             // Bandwidth for sweep
    double fdel;                    // Frequency step for sweep
    double fstart;                          // Start frequency for sweep
    double fstop;                   // Stop frequency for sweep
    double tdel;                            // Step time for sweep [ms]

    parini();                               // Initialize parallel port
    fstart = 1.0E+9;                        // Initialize sweep parameters
    fstop = 2.0E+9;
    fdel = 10.0E+6;
    tdel = 500.0;
    printf("TESTHSC %s\n",VERSION);         // Output startup message

    for (;;) {
        stat = (unsigned long) inp(HAND_IN);            // Get handshake in
        printf("Enter command[");                       //  and display with the
        if (TEST(stat,NREADY)) printf("Not ready,");    //  prompt
        else printf("Ready,");
```

139

```c
if (TEST(stat,SYNC)) printf("Sync");
else printf("Not sync");
printf("]?");

for (i=0;(i<79)&&((ch=getchar())!='\n');i++) str[i]=(char)ch; // Get cmd
str[i] = '\0';
nscan = sscanf(str,"%s %lx",comd,&parm);          // Break to comd & value
if (i == 0) {
    hop();                                        // No comd => hop
} else if (strcmpi(comd,"S")==0) {
    ;    // do nothing, get next status          // 'S' => redo status
} else if ((strcmpi(comd,"HELP")==0) || (strcmpi(comd,"H")==0) ||
           (strcmpi(comd,"?")==0)) {              // 'HELP' help
    printf("\nTESTHSC  %s                        "
           "           Use HELP or VAL for help\n",VERSION);
    printf("Tests the Hopping Synthesizer Controller using the "
           "DT2817 Parallel Card\n\n");
    printf("  General Commands                   |"
           "    Load Commands\n");
    printf("RUN                 Run Mode         |"
           " LOAD_LATCH #    Synth latch directly\n");
    printf("STOP                Stop Mode        |"
           " LOAD_SEED #     Random number seed\n");
    printf("DEFAULTS            Reinit parameter |"
           " LOAD_HOP #      Hop number then latch\n");
    printf("CHANGE_HOP          Latch on Hop Clock |"
           " LOW,MID,HIGH    Special cases LOAD_HOP\n");
    printf("CHANGE_IMMEDIATE    Latch immediate  |"
           " LOAD_BWSCALE #  BW scale [200 Hz]\n");
    printf("READ <addr>         Read parameter   |"
           " LOAD_FSKCHAN #  FSK/Channel\n");
    printf("HOP or <CR>         Pulse Hop Clock  |"
           " LOAD_FCSPACE #  Tone spacing [100 Hz]\n");
    printf("SWEEP [0]           Sweep (0=new vals) |"
           " LOAD_BASE #     Base frequency [100 Hz]\n");
    printf("QUIT                Exit             |"
           " LOAD_OFFSET #   Offset freq [100 Hz]\n");
    printf("  Sync Commands                      |"
           " LOAD_DOPFACT #  Doppler factor\n");
    printf("DLSYNC              D/L Sync Mode    |"
           " LOAD_LOCOM #    Comst LO freq [100 Hz]\n");
    printf("DLGO 0-4            D/L Sync Hop Go   |"
           " LOAD_TIMELO #   Time low word [hops]\n");
    printf("ULSYNC <fskchn>     U/L Sync Mode    |"
           " LOAD_TIMEHI #   Time high word [hops]\n");
    printf("FCALC               U/L Precalculate |"
           " LOAD_LOSCI #    Sciteq LO freq [100 Hz]\n");
    printf("ULGO 0-1F           U/L Sync Probe Go |"
           " LOAD_FLAGS #    Flags (1=/2, 2=run)\n");
} else if ((strcmpi(comd,"VALUE")==0) || (strcmpi(comd,"VAL")==0)) {
    printf("\nCommand Values (hex)              "     // 'VAL' values
           "           Use HELP or VAL for help\n\n");
    printf("  General Commands                   |"
           "    Load Commands\n");
    printf("RUN                 8000             |"
           " LOAD_LATCH #       0300\n");
    printf("STOP                0000             |"
           " LOAD_SEED #        0302\n");
    printf("DEFAULTS            0400             |"
           " LOAD_HOP #         0304\n");
    printf("CHANGE_HOP          0500             |"
           " LOW,MID,HIGH       (0,800000,FFFFFF)\n");
    printf("CHANGE_IMMEDIATE    0501             |"
           " LOAD_BWSCALE #     0306\n");
    printf("READ <addr>         2000-2FFF        |"
           " LOAD_FSKCHAN #     0308\n");
    printf("HOP or <CR>         -                |"
           " LOAD_FCSPACE #     030A\n");
```

140

```
        printf("SWEEP            -              !"
               " LOAD_BASE #          030C\n");
        printf("QUIT             -              !"
               " LOAD_OFFSET #        030E\n");
        printf("  Sync Commands                 !"
               " LOAD_DOPFACT #       0310\n");
        printf("DLSYNC           4000           !"
               " LOAD_LOCOM #         0312\n");
        printf("DLGO 0-4         60,62..68      !"
               " LOAD_TIMELO #        0314\n");
        printf("ULSYNC <fskchn>  5000-50FF      !"
               " LOAD_TIMEHI #        0316\n");
        printf("FCALC            5800           !"
               " LOAD_LOSCI #         0318\n");
        printf("ULGO 0-1F        200,202..23E   !"
               " LOAD_FLAGS #         031A\n");
} else if (strcmpi(comd,"DLGO")==0) {          // Downlink GO
    if (nscan == 2) {
            if (parm <=4) {
                intout((unsigned long) MDLGO+parm*2);
            } else {
                printf(" Valid frequency numbers for DLGO are 0-4.\n");
                printf("  0-3 for Hops 0-3 of Burst 0.\n");
                printf("  4   for Hop 0 of Burst 1.\n");
            }
    } else {
            printf(" DLGO requires a frequency number (0-4).\n");
                printf("  0-3 for Hops 0-3 of Burst 0.\n");
                printf("  4   for Hop 0 of Burst 1.\n");
    }
} else if (strcmpi(comd,"ULGO")==0) {          // Uplink GO
    if (nscan == 2) {
            if (parm < 32) {
                intout((unsigned long) MULGO+parm*2);
            } else {
                printf(" Valid frequency numbers for ULGO are 0-1F.\n");
                printf("  0-F  for Probes 0-15 of Burst 0.\n");
                printf("  10-1F for Probes 0-15 of Burst 1.\n");
            }
    } else {
            printf(" ULGO requires a frequency number (0-1F).\n");
                printf("  0-F  for Probes 0-15 of Burst 0.\n");
                printf("  10-1F for Probes 0-15 of Burst 1.\n");
    }
} else if (strcmpi(comd,"FCALC")==0) {         // Frame calculation
    if (nscan == 1) {
            intout((unsigned long) MFCALC);
    } else {
            printf(" FCALC does not take any parameters.\n");
    }
} else if (strcmpi(comd,"RUN")==0) {           // Switch to RUN Mode
    if (nscan == 1) {
            intout((unsigned long) MRUN);
    } else {
            printf(" RUN does not take any parameters.\n");
    }
} else if (strcmpi(comd,"STOP")==0) {          // Switch to STOP Mode
    if (nscan == 1) {
            intout((unsigned long) MSTOP);
    } else {
            printf(" STOP does not take any parameters.\n");
    }
} else if (strcmpi(comd,"DLSYNC")==0) {        // Switch to DLSYNC Mode
    if (nscan == 1) {
            intout((unsigned long) MDLSYN);
    } else {
            printf(" DLSYNC does not take any parameters.\n");
    }
```

```
    } else if (strcmpi(comd,"ULSYNC")==0) {          // Switch to ULSYNC Mode
        if (nscan == 2) {
            intout((unsigned long) MULSYN+parm);
        } else {
            printf(" ULSYNC requires hex value (0-FF) for FSK/Channel.\n");
            printf("  bits are <cccccfff> where fff is tone 0-7\n");
            printf("                           ccccc is channel 0-31\n");
        }
    } else if ((strcmpi(comd,"SETDEF")==0) ||         // Select defaults
                (strcmpi(comd,"DEFAULTS") == 0)) {
        if (nscan == 1) {
            intout((unsigned long) MDEFLT);
        } else {
            printf(" DEFAULTS does not take any parameters.\n");
        }
    } else if ((strcmpi(comd,"RXF")==0) ||            // Change on hop
                (strcmpi(comd,"CHANGE_HOP")==0)) {
        if (nscan == 1) {
            intout((unsigned long) MCGHOP);
        } else {
            printf(" CHANGE_HOP does not take any parameters.\n");
        }
    } else if ((strcmpi(comd,"SXF")==0) ||            // Change immediately
                (strcmpi(comd,"CHANGE_IMMEDIATE")==0)) {
        if (nscan == 1) {
            intout((unsigned long) MCGIMM);
        } else {
            printf(" CHANGE_IMMEDIATE does not take any parameters.\n");
        }
    } else if (strcmpi(comd,"LOAD_LATCH")==0) {        // Load latches
        if (nscan == 2) {
            loadout((unsigned long) MLDLTC, parm);
        } else {
            printf(" LOAD_LATCH requires a hex value (0-FFFFFFFF).\n");
            printf("  The value is loaded directly into latches.\n");
        }
    } else if (strcmpi(comd,"LOAD_SEED")==0) {         // Load random seed
        if (nscan == 2) {
            loadout((unsigned long) MLDSED,parm);
        } else {
            printf(" LOAD_SEED requires a hex value (0-7FFFFFFF).\n");
        }
    } else if (strcmpi(comd,"LOAD_HOP")==0) {          // Load hop number
        if (nscan == 2) {
            loadout((unsigned long) MLDHOP,parm);
        } else {
            printf(" LOAD_HOP requires a hex value (0-FFFFFF).\n");
            printf("  The value specified is the hop number within BW.\n");
        }
    } else if (strcmpi(comd,"LOW")==0) {               // Load hop number min
        if (nscan == 1) {
            loadout((unsigned long) MLDHOP,0x00000000L);
        } else {
            printf(" LOW does not take any parameters.\n");
        }
    } else if (strcmpi(comd,"MID")==0) {               // Load hop number mid
        if (nscan == 1) {
            loadout((unsigned long) MLDHOP,0x00800000L);
        } else {
            printf(" MID does not take any parameters.\n");
        }
    } else if (strcmpi(comd,"HIGH")==0) {              // Load hop number max
        if (nscan == 1) {
            loadout((unsigned long) MLDHOP,0x00FFFFFFL);
        } else {
            printf(" HIGH does not take any parameters.\n");
        }
    } else if (strcmpi(comd,"LOAD_BWSCALE")==0) {      // Load bandwidth scale
```

142

```
                    printf("  The value is the Sciteq LO frequency [100 Hz].\n");
          }
     } else if (strcmpi(comd,"LOAD_FLAGS")==0) {        // Load config flags
          if (nscan == 2) {
                    loadout((unsigned long) MLDFLG,parm);
          } else {
                    printf(" LOAD_FLAGS requires a hex value (0-3).\n");
                    printf("  bits are <rd> where r is startup run mode, 1=yes\n");
                    printf("                          d is Sciteq divide by 2,1=yes\n");
          }
     } else if (strcmpi(comd,"READ")==0) {              // Read TMS RAM
          if (nscan == 2) {
                    intout((unsigned long) MREAD+parm);        // Lower 16 bits
                    val_lo = intin();
                    intout((unsigned long) MREAD+parm+1);      // Upper 16 bits
                    val_hi = intin();
                    val = (val_hi<<16) + val_lo;               // Make 32 bits
                    printf("  (%lX,%lX) = %8lXh   [%ld]\n",parm,parm+1,val,val);
          } else {
                    printf(" READ requires a hex address (0-FFF).\n");
          }
     } else if (strcmpi(comd,"HOP")==0) {               // 'HOP' => hop
        hop();
     } else if (strcmpi(comd,"SWEEP")==0) {             // 'SWEEP'
          if (nscan == 2) {                             // parm => init
                    printf("  Enter start frequency [GHz]?");
                    scanf("%lg",&fstart);
                    fstart *= 1.0E+9;
                    printf("  Enter stop frequency [GHz]?");
                    scanf("%lg",&fstop);
                    fstop *= 1.0E+9;
                    printf("  Enter step size [GHz]?");
                    scanf("%lg",&fdel);
                    fdel *= 1.0E+9;
                    printf("  Enter time per step [ms]?");
                    scanf("%lg",&tdel);
                    getchar();
          }
          if ((fstart < fstop) && (fstop > 0) && ((fstop-fstart) > fdel) &&
                    (tdel < 2000.0) && (tdel >= 1.0)) {
                    ntick = tdel + 0.5;                         // Calc # ticks/step
                    fbw = fstop - fstart;
                    printf("  Overwriting FB,SCP,FCD,FOF,DPF,FLO,LOS\n");
                    loadout((unsigned long) MLDBAS,(unsigned long) fstart/100);
                    loadout((unsigned long) MLDBWS,(unsigned long) ((fbw/2)/100));
                    loadout((unsigned long) MLDFCS,(unsigned long) 0);
                    loadout((unsigned long) MLDOFF,(unsigned long) 0);
                    loadout((unsigned long) MLDDPF,(unsigned long) 0);
                    loadout((unsigned long) MLDLOC,(unsigned long) 0);
                    loadout((unsigned long) MLDLOS,(unsigned long) 0);
                    intout((unsigned long) MCGIMM);            // Change immediately

                    printf("  Sweeping %lg to %lg step %lg [Hz] ",
                              fstart,fstop,fdel);
                    for (f=fstart;f<fstop;f+=fdel) {           // Do sweep
                         lf = (((f-fstart)/fbw)*0x1000000L + 0.5);
                         loadout((unsigned long) MLDHOP,lf);
                         if (wait(ntick) == 0) {               // Abort on key press
                           printf("*** Key Abort *** ");
                           getch();
                           break;
                         }
                    }
                    intout((unsigned long) MCGHOP);            // Change on hop
                    printf(" Done\n");
          } else {
                    printf("  Error in parameters, the following must be met:\n");
                    printf("     stop > start, stop > 0, start > 0\n");
```

144

```
                printf("     step < stop-start\n");
                printf("     time < 2.0 s, time >= 1 ms\n");
            }
        } else if (strcmpi(comd,"QUIT")==0) {            // Quit and aliases
            exit(0);
        } else if (strcmpi(comd,"Q")==0) {
            exit(0);
        } else if (strcmpi(comd,"EXIT")==0) {
            exit(0);
        } else if (strcmpi(comd,"E")==0) {
            exit(0);
        } else if (strcmpi(comd,"LOAD") == 0) {           // Hint for loads
            printf(" LOAD commands require an underscore (ex: LOAD_BASE 10)\n");
        } else if (strcmpi(comd,"CHANGE") == 0) {         // Hint for changes
            printf(" CHANGE commands require an underscore (ex: CHANGE_HOP)\n");
        } else {
            printf(" Illegal command, type HELP for valid commands.\n");
        }
    }
}


//=============================================================================
//
//        intin - Read a 16-bit value from the HSC
//
//-----------------------------------------------------------------------------
//
//        In:             -
//        Out:            -
//        Return:         unsigned long        16-bit value read from the
//                                             address in the previous READ
//                                             command
//
//-----------------------------------------------------------------------------

unsigned long intin(void)
{
    int i;                      // Integer index
    unsigned long value;  // Value read
    unsigned long stat;         // Status of handshake in
    unsigned long bit;          // Bit read

    value = 0;
    for (i=0;i<16;i++) {
            intout((unsigned long) MREAD+15-i);      // Output READ to clock out bit
                                                     // (actual value doesn't matter)
            stat = (unsigned long) inp(HAND_IN);     // Get handshake lines
            bit = (stat >> SYNC) & 0X01;             // Extract SYNC bit for data
            value = (value << 1) + bit;              // Save bit
    }
    intout((unsigned long) MSTOP);                   // Output STOP twice to end READ
    intout((unsigned long) MSTOP);
    return(value);
}


//=============================================================================
//
//        hop - Generate a pulse on the hop clock line
//
//-----------------------------------------------------------------------------
//
//        In:             -
//        Out:            -
//        Return:         -
//
//-----------------------------------------------------------------------------
```

145

```c
void hop(void)
{
    shadow = SET(shadow,HCLK);              // Set hop clock bit in shadow
    (void) outp(HAND_OUT,shadow);           // Output to handshake out
    shadow = RESET(shadow,HCLK);        // Clear hop clock bit
    (void) outp(HAND_OUT,shadow);           // Output to handshake out
    return;
}
```

```
//===============================================================================
//
//         intout - Output a 16-bit value with strobe to the HSC
//
//-------------------------------------------------------------------------------
//
//         In:            unsigned long ival      16-bit value to be sent to HSC
//         Out:           -
//         Return:        -
//
//-------------------------------------------------------------------------------
```

```c
void intout(unsigned long ival)
{
    if (ready() == 0) {                              // Wait for HSC ready
        printf(" Interface not ready in intout.\n");
    } else {
        (void) outp(LOBYTE,(int) (ival & OXFF));      // Set low 8 bits
        (void) outp(HIBYTE,(int) ((ival >> 8) & OXFF)); // Set high 8 bits
        strobe();                                      // Pulse strobe line
    }
    return;
}
```

```
//===============================================================================
//
//         loadout - Ouput a load command with 32-bit parameter
//
//-------------------------------------------------------------------------------
//
//         In:            unsigned long cmd       Load command to be sent to HSC
//                        unsigned long lval       32-bit value for load command
//         Out:           -
//         Return:        -
//
//-------------------------------------------------------------------------------
```

```c
void loadout(unsigned long cmd,unsigned long lval)
{
    intout(cmd);                    // Send load command
    longout(lval);                       // Send 32-bit parameter
    return;
}
```

```
//===============================================================================
//
//         longout - Output a 32-bit value to the HSC
//
//-------------------------------------------------------------------------------
//
//         In:            unsigned long lval      32-bit value to be sent to HSC
//         Out:           -
//         Return:        -
//
//-------------------------------------------------------------------------------
```

```c
void longout(unsigned long lval)
{
    intout(lval & 0X0FFFF);                 // Send lower 16 bits
    intout((lval >> 16) & 0X0FFFF);         // Send upper 16 bits
    return;
}


//===============================================================================
//
//        parini - Initialize the parallel port card
//
//-------------------------------------------------------------------------------
//
//        In:            -
//        Out:           -
//        Return:        -
//
//-------------------------------------------------------------------------------

void parini(void)
{
    (void) outp(CONTRL,0X07);               // Set Ports 0-2 output, Port 3 input
    (void) outp(LOBYTE,0X00);               // Clear data out registers
    (void) outp(HIBYTE,0X00);
    shadow = 0X00;                          // Clear shadow register
    (void) outp(HAND_OUT,shadow);           // Clear handshake out register
    return;
}


//===============================================================================
//
//        ready - Wait for HSC ready (when the previous command is done)
//
//-------------------------------------------------------------------------------
//
//        In:            -
//        Out:           -
//        Return:        int        Number of loops until ready
//                                   (0 if not ready timeout)
//
//-------------------------------------------------------------------------------

int ready(void)
{
    int i;                          // Integer index
    unsigned long stat;             // Status of handshake in

    i = 0;                                  // Initialize loop counter
    do {
        stat = (unsigned long) inp(HAND_IN);    // Get handshake in status
        i++;
        if (i > 10000) return i;            // Limit number of loops
    } while (!TEST(stat,NREADY));           // Loop until NReady clear
    return 0;
}


//===============================================================================
//
//        strobe - Generate a pulse on the strobe line
//
//-------------------------------------------------------------------------------
//
//        In:            -
//        Out:           -
//        Return:        -
```

147

```
//
//-------------------------------------------------------------------

void strobe(void)
{
    shadow = SET(shadow,STROBE);        // Set strobe bit in shadow
    (void) outp(HAND_OUT,shadow);            // Output to handshake out
    shadow = RESET(shadow,STROBE);           // Clear strobe bit in shadow
    (void) outp(HAND_OUT,shadow);            // Output to handshake out
    return;
}



//===============================================================================
//
//      wait - Wait for specified number of clock ticks
//
//-------------------------------------------------------------------
//
//      In:         int nticks      Number of 60 ms ticks to wait
//      Out:        -
//      Return:     int             -1 if processor time not available
//                                   0 if a key has been pressed
//                                   1 if wait sucessfully completed
//
//-------------------------------------------------------------------

int wait(int nticks)
{
    long beg_time;              // Number of ticks since 0 for start time
    long dif_time;              // Number of ticks between start time and now

    if ((beg_time=clock()) == (clock_t) -1) {          // Get start time
        printf(" Processor time not available\n");
        exit(-1);
    }
    dif_time = clock() - beg_time;                  // Get difference
    while (dif_time < nticks) {                      // Wait until diff is
        dif_time = clock() - beg_time;               //  greater than nticks
        if (kbhit() != 0) return 0;                  // Abort on key press
    }
    return 1;
}
```

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence Research Establishment Ottawa Ottawa, Ontario K1A 0Z4	**2. SECURITY CLASSIFICATION** (overall security classification of the document including special warning terms if applicable) **UNCLASSIFIED**

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Modified Hopping Synthesizer Controller (U)

4. AUTHORS (Last name, first name, middle initial)

Addison, Robin D.

5. DATE OF PUBLICATION (month and year of publication of document) December 1996	**6a. NO. OF PAGES** (total containing information. Include Annexes, Appendices, etc.) 164	**6b. NO. OF REFS** (total cited in document) 12

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.

SST, Defence Research Establishment Ottawa
Ottawa, Ontario, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant) 5ca11	**9b. CONTRACT NO.** (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) DREO REPORT 1304	**10b. OTHER DOCUMENT NOS.** (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). however, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

-150-

7

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Defence Research Establishment Ottawa is pursuing an in-house research activity in spread-spectrum technology to support development of robust, anti-jam satellite communications for the military. The in-house effort consists of developing a system simulator to research the techniques involved in spread-spectrum synchronization. For these experiments, a Hopping Synthesizer Controller is required to drive a frequency synthesizer, and is the subject of this report. The controller was built on a single board using a digital signal processor for calculations and an erasable programmable logic device for interfacing. It provides the pseudo-random hopping sequence, modulation, channel selection and Doppler precorrection. The Hopping Synthesizer Controller can drive the frequency synthesizer at hop rates up to 25 khops/s. Four configurations are supported: operation within either the payload or ground terminal simulator, for uplink or downlink. Each configuration supports commands from a Synchronization Controller and special uplink and downlink synchronization modes. In this document the software and hardware details are provided along with a user guide for the board.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

spread spectrum
synthesizer
frequency hopping
satellite communications

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM